Information and Computation 209 (2011) 799–821

Contents lists available at ScienceDirect

Information and Computation

journal homepage: www.elsevier.com/locate/ic

A saturation method for the modal μ-calculus over pushdown systems

M. Hague *, C.-H.L. Ong

Oxford University Computing Laboratory, Wolfson Building, Parks Road, Oxford OX1 3QD, UK

ARTICLE INFO

Article history:
Available online 15 December 2010

Keywords:
Modal μ-calculus
Pushdown systems
Parity games
Winning regions
Global model checking
Saturation methods

ABSTRACT

We present an algorithm for computing *directly* the denotation of a modal μ-calculus formula χ over the configuration graph of a pushdown system. Our method gives the first extension of the saturation technique to the full modal μ-calculus. Finite word automata are used to represent sets of pushdown configurations. Starting from an initial automaton, we perform a series of automaton manipulations which compute the denotation by recursion over the structure of the formula. We introduce notions of under-approximation (soundness) and over-approximation (completeness) that apply to automaton transitions rather than runs. Our algorithm is relatively simple and direct, and avoids an immediate exponential blow up. Finally, we show experimentally that the direct algorithm is more efficient than via a reduction to parity games.

1. Introduction

Pushdown systems – finite-state transition systems equipped with a stack – are an old model of computation that have recently enjoyed renewed interest from the software verification community. They accurately model the control flow of first-order recursive programs [18] (such as C and Java), and lend themselves readily to algorithmic analysis. Pushdown systems have played a key rôle in the automata-theoretic approach to software model checking [6,13,20,21]. Considerable progress has been made in the implementation of scalable model checkers of pushdown systems. These tools (e.g., Bebop [3] and Moped [21]) are an essential back-end component of such model checkers as SLAM [4].

The modal μ-calculus is a highly expressive language for describing properties of program behaviour (all standard temporal logics in verification are embeddable in it). In a seminal paper [23] at CAV 1996, Walukiewicz showed that *local* modal μ-calculus model checking of pushdown systems – or equivalently [12] the solution of *pushdown parity games* (i.e., parity games over the configuration graphs of pushdown systems) – is EXPTIME-complete. His method reduces pushdown parity games to finite parity games by a kind of powerset construction, which is immediately exponential in size.

Whilst *local* model checking asks if a designated state (of a pushdown system) satisfies a given property, *global* model checking computes a finite representation of the set of states satisfying the property. It is worth noting that global model checking used to be the norm in verification (CTL and many symbolic model checkers still perform global model checking). While local model checking can be expected to have better complexity, global model checking is important when repeated checks are required (because tests on the representing automata tend to be comparatively cheap), or where the model checking is only a component of the verification process.

1.1. Contributions

This paper presents a new algorithm for solving the global model checking problem for modal μ-calculus over pushdown systems. That is, given a pushdown system $\mathbb{P}$, a modal μ-calculus formula $\chi(\overline{Z})$ for $\overline{Z} = Z_1, \ldots, Z_n$, and a regular valuation V,

* Corresponding author. Fax: +44 1865 273839.
E-mail addresses: Matthew.Hague@comlab.ox.ac.uk (M. Hague), Luke.Ong@comlab.ox.ac.uk (C.-H.L. Ong).

0890-5401/$ - see front matter © 2010 Elsevier Inc. All rights reserved.
doi:10.1016/j.ic.2010.12.004

our method can *directly* compute an automaton that recognises the set $[\![\chi(\overline{Z})]\!]^{\mathbb{P}}_V$ of $\mathbb{P}$-configurations satisfying $\chi(\overline{Z})$ with respect to V.

We represent the (regular) configuration sets as alternating multi-automata [6]. To evaluate a fixed point formula, our algorithm iteratively expands (when computing least fixed points) or contracts (when computing greatest fixed points) an approximating automaton until the denotation is precisely recognised. Our method is a generalisation of Cachat's for solving Büchi games [10,11], which is itself a generalisation of the saturation technique for reachability analysis [6]. A specialised version of this algorithm was presented in Concur 2009 [16]. This simplified algorithm computes, using a modal μ-calculus formula as a guide, the winning regions of a pushdown parity game.

Our algorithm has several advantages:

1. The algorithm is relatively simple and direct. Even though pushdown graphs are in general infinite, our construction of the automaton that recognises the denotation follows, in outline, the standard pen-and-paper calculation of the semantics of modal μ-calculus formulas in a *finite* transition system. Through the use of *projection*, our algorithm is guaranteed to terminate in a finite number of steps, even though the usual fixed point calculations may require transfinite iterations. Thanks to projection, the state-sets of the approximating automata are bounded: during expansion, the number of transitions increases, but only up to the bound determined by the finite state-set; during contraction, the number of transitions decreases until it reaches zero or stabilizes.
2. Conceptual innovations of the correctness argument are *valuation soundness* and *valuation completeness*. They are, respectively, under- and over-approximation conditions that apply *locally* to individual transitions of the automaton, rather than *globally* to the extensional behaviour of the automaton (such as runs). By combining these conditions, which reduce the overhead of the proof,[1] we show that our algorithm is both sound and complete in the usual sense.
3. The algorithm, in essence, combines the product construction – that reduces a modal μ-calculus model checking problem to a pushdown parity game – and the computation of the winning region. However, this direct computation only introduces product states that are relevant to the evaluation of the current *sub-formula* (rather than the whole formula), hence the number of states used is minimised. Since the algorithm is exponential in the number of states, even a slight reduction in the number of states can lead to significant improvements in run-times. We confirm this experimentally in Section 9.
4. Finally, our decision procedure builds on and extends the well-known saturation method, which is the implementation technique of choice of pushdown checkers. In contrast to previous solutions, our algorithm avoids an immediate exponential explosion, which we believe is important for an efficient implementation.

1.2. Related work

Cachat [11] and Serre [22] have independently generalised Walukiewicz' algorithm to provide solutions to the global model-checking problem: they use the local model-checking algorithm as an oracle to guide the construction of the automaton recognising the winning region. An alternative approach, introduced by Piterman and Vardi [19], uses two-way alternating tree automata to navigate a tree representing all possible stacks: after several reductions, including the complementation of Büchi automata, an automaton accepting the winning regions can be constructed.

An early technique for analysing modal μ-calculus properties of pushdown systems is due to Burkart and Steffen [9]. They provide an algorithm for analysing context-free systems by reduction to a finite equational fixed point computation. This can be extended to pushdown systems by adding arguments to the equations, and then performing a computation argument-wise for each of the exponential number of arguments [8].

At Concur 1997, Bouajjani et al. [6], and, independently, Finkel et al. [15] (at Infinity 1997), introduced a *saturation* technique for global model-checking reachability properties of pushdown systems. This technique was based on a string-rewriting algorithm due to Book and Otto [5]. From a finite-word automaton recognising a given configuration-set $\mathcal{C}$, they perform a backwards-reachability analysis. By iteratively adding new transitions to the automaton, the set of configurations that can reach some configuration in $\mathcal{C}$ is constructed. Since the number of new transitions is bounded, the iterative process terminates. This approach underpins the acclaimed Moped tool.

The saturation technique was generalised by Cachat to compute the winning regions of Büchi games [10]. By using *projections*, Cachat was able to show how to compute a single alternation of fixed points. We have generalised this approach to compute an arbitrary number of fixed points. Furthermore, we believe that the introduction of valuation-soundness and -completeness leads to a cleaner proof of correctness. An "automaton free" version of Cachat's approach was given by Etessami [14]. This approach computes the winning regions of a Büchi game using data flow equations. To our knowledge, it has not been applied to parity games, although such an extension may be possible.

Finally, Alur et al. introduce a version of the modal μ-calculus for recursive programs [1]. This logic is more expressive than the modal μ-calculus. A investigation of further properties, such as succinctness, is an interesting avenue of future work.

[1] Although some proofs are long, this is primarily due to the number of cases involved in an induction over the syntax of the modal μ-calculus. The proofs themselves are fairly straightforward.

2. Preliminaries

2.1. Pushdown systems

A ***pushdown system*** (PDS) is a triple $\mathbb{P} = (\mathcal{P}, \mathcal{D}, \Sigma_\perp)$ where $\mathcal{P}$ is a set of control states, $\Sigma_\perp := \Sigma \cup \{\perp\}$ is a finite stack alphabet (we assume $\perp \notin \Sigma$), $\mathcal{D} \subseteq \mathcal{P} \times \Sigma_\perp \times \mathcal{P} \times \Sigma_\perp^*$ is a set of pushdown rules. As is standard, we assume that the bottom-of-stack symbol $\perp$ is neither pushed onto, nor popped from, the stack. We write $\langle p, aw\rangle \hookrightarrow \langle p', w'w\rangle$ whenever $p\,a \to p'\,w' \in \mathcal{D}$ and $\mathcal{C}$ to refer to the set of all pushdown configurations.

2.2. Modal μ-Calculus

Given a set of propositions AP and a disjoint set of variables $\mathcal{Z}$, formulas of the modal μ-calculus are defined as follows (with $x \in AP$ and $Z \in \mathcal{Z}$):

$$\varphi \;:=\; x \mid \neg x \mid Z \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \Box\varphi \mid \Diamond\varphi \mid \mu Z.\varphi \mid \nu Z.\varphi.$$

Thus we assume that the formulas are in *positive form*, in the sense that negation is only applied to atomic propositions. Over a pushdown system, the semantics of a formula φ are given with respect to a *valuation* $V : \mathcal{Z} \to \mathcal{P}(\mathcal{C})$ which maps each free variable to its set of satisfying configurations and an environment $\rho : AP \to \mathcal{P}(\mathcal{C})$ mapping each atomic proposition to its set of satisfying configurations. We then have,

$$\begin{aligned}
[\![x]\!]^{\mathbb{P}}_V &= \rho(x)\\
[\![\neg x]\!]^{\mathbb{P}}_V &= \mathcal{C} \setminus \rho(x)\\
[\![Z]\!]^{\mathbb{P}}_V &= V(Z)\\
[\![\varphi_1 \wedge \varphi_2]\!]^{\mathbb{P}}_V &= [\![\varphi_1]\!]^{\mathbb{P}}_V \cap [\![\varphi_2]\!]^{\mathbb{P}}_V\\
[\![\varphi_1 \vee \varphi_2]\!]^{\mathbb{P}}_V &= [\![\varphi_1]\!]^{\mathbb{P}}_V \cup [\![\varphi_2]\!]^{\mathbb{P}}_V\\
[\![\Box\varphi]\!]^{\mathbb{P}}_V &= \left\{ c \in \mathcal{C} \;\middle|\; \forall c'.c \hookrightarrow c' \Rightarrow c' \in [\![\varphi]\!]^{\mathbb{P}}_V \right\}\\
[\![\Diamond\varphi]\!]^{\mathbb{P}}_V &= \left\{ c \in \mathcal{C} \;\middle|\; \exists c'.c \hookrightarrow c' \wedge c' \in [\![\varphi]\!]^{\mathbb{P}}_V \right\}\\
[\![\mu Z.\varphi]\!]^{\mathbb{P}}_V &= \bigcap \left\{ S \subseteq \mathcal{C} \;\middle|\; [\![\varphi]\!]^{\mathbb{P}}_{V[Z\mapsto S]} \subseteq S \right\}\\
[\![\nu Z.\varphi]\!]^{\mathbb{P}}_V &= \bigcup \left\{ S \subseteq \mathcal{C} \;\middle|\; S \subseteq [\![\varphi]\!]^{\mathbb{P}}_{V[Z\mapsto S]} \right\}
\end{aligned}$$

where $V[Z \mapsto S]$ updates the valuation V to map the variable Z to the set S.

The operators $\Box\varphi$ and $\Diamond\varphi$ assert that φ holds after all possible transitions and after some transition, respectively; and the μ and ν operators specify greatest and least fixed points. Another interpretation of these operators is given below. For a full discussion of the modal μ-calculus we refer the reader to a survey by Bradfield and Stirling [7].

2.3. Approximants

Thanks to the Knaster–Tarski Fixed Point Theorem, the semantics of a fixed point formula $[\![\sigma Z.\chi(\overline{Y}, Z)]\!]^{\mathbb{P}}_V$ where $\overline{Y} = Y_1, \ldots, Y_n$ and $\sigma \in \{\mu, \nu\}$ can be given as the limit of the sequence of ***α-approximants*** $[\![\sigma^\alpha Z.\chi(\overline{Y}, Z)]\!]^{\mathbb{P}}_V$, where α ranges over the ordinals and λ ranges over the limit ordinals:

$$\begin{aligned}
[\![\sigma^0 Z.\chi(\overline{Y}, Z)]\!]^{\mathbb{P}}_V &:= \mathit{Init}\\
[\![\sigma^{\alpha+1} Z.\chi(\overline{Y}, Z)]\!]^{\mathbb{P}}_V &:= [\![\chi(\overline{Y}, Z)]\!]^{\mathcal{G}}_{V[Z\mapsto [\![\sigma^\alpha Z.\chi(\overline{Y},Z)]\!]^{\mathbb{P}}_V]}\\
[\![\sigma^\lambda Z.\chi(\overline{Y}, Z)]\!]^{\mathbb{P}}_V &:= \textstyle\bigcirc_{\alpha<\lambda} [\![\sigma^\alpha Z.\chi(\overline{Y}, Z)]\!]^{\mathbb{P}}_V
\end{aligned}$$

where $\mathit{Init} = \emptyset$ and $\bigcirc = \bigcup$ when $\sigma = \mu$, and Init is the set of all configurations and $\bigcirc = \bigcap$ when $\sigma = \nu$. The least ordinal κ such that $[\![\sigma^\kappa Z.\chi(\overline{Y}, Z)]\!]^{\mathbb{P}}_V = [\![\sigma Z.\chi(\overline{Y}, Z)]\!]^{\mathbb{P}}_V$ is called the *closure ordinal*.

Example 2.1. When interpreted in a pushdown graph, $[\![\sigma^\alpha Z.\chi(\overline{Y}, Z)]\!]_{\alpha\in\mathbf{Ord}}$ may have a closure ordinal strictly greater than ω. Consider the pushdown graph in Fig. 1 (which is a dual of an example of Cachat's [11]). The graph is generated by the pushdown system with the rules

$$\begin{array}{ll}
p \perp \to f \perp & f \perp \to f \perp\\
p\,a \to p & f\,a \to f\,a\,a\\
 & f\,a \to p\,a.
\end{array}$$

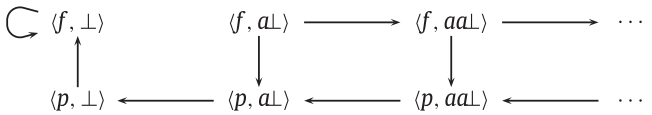

Fig. 1. The configuration graph of an example pushdown system.

The proposition p is true only when the control state is p and f is true only at control state f. In this graph $[\![\mu Z_1.\nu Z_2.\ (p \wedge \Box Z_1) \vee (f \wedge \Box Z_2)]\!]$ consists of all configurations. However, any $\langle f, a\, a^n\bot\rangle$ for some n only appears in an approximant of the least fixed point when $\langle f, a\, a\, a^n\bot\rangle$ and $\langle p, a\, a^n\bot\rangle$ appear in the previous approximant (since $\Box Z_2$ quantifies over all transitions from $\langle f, a\, a^n\bot\rangle$). Hence, all $\langle p, a^n\bot\rangle$ must appear in the α-approximant before any $\langle f, a^n\bot\rangle$ can appear in the $(\alpha + 1)$-approximant. Thus the first approximant containing all p configurations is the ω-approximant. It follows that the least fixed point in question has a closure ordinal larger than ω. Cachat also shows that the same holds for greatest fixed points.

2.4. Alternating multi-automata

We use alternating multi-automata [6] as a representation of (regular) sets of configurations. Given a pushdown system $(\mathcal{P}, \mathcal{D}, \Sigma_\bot)$ with $\mathcal{P} = \{p^1, \ldots, p^z\}$, an ***alternating multi-automaton*** A is a quintuple $(\mathcal{Q}, \Sigma_\bot, \Delta, I, \mathcal{F})$ where $\mathcal{Q}$ is a finite set of states, $\Delta \subseteq \mathcal{Q} \times (\Sigma \cup \{\bot\}) \times 2^{\mathcal{Q}}$ is a set of transitions (we assume $\bot \notin \Sigma$), $I = \{q^1, \ldots, q^z\} \subseteq \mathcal{Q}$ is a set of initial states, and $\mathcal{F} \subseteq \mathcal{Q}$ is a set of final states. Observe that there is an initial state for each control state of the pushdown system. We write $q \xrightarrow{a} Q$ just if $(q, a, Q) \in \Delta$; and define $q \xrightarrow{\varepsilon} \{q\}$; and $q \xrightarrow{aw} Q_1 \cup \cdots \cup Q_n$ just if $q \xrightarrow{a} \{q_1, \ldots, q_n\}$ and $q_k \xrightarrow{w} Q_k$ for all $1 \leq k \leq n$. Finally we define the *language accepted by A*, $\mathcal{L}(A)$, by: $\langle p^j, w\rangle \in \mathcal{L}(A)$ just if $q^j \xrightarrow{w} Q$ for some $Q \subseteq \mathcal{F}$. We further define $\mathcal{L}_q(A)$ to be the set of all words accepted from the state q in A. Henceforth, we shall refer to alternating multi-automata simply as ***automata***. In cases of ambiguity, we may specify runs of a particular automaton A with a transition relation Δ by $q \xrightarrow[A]{a} Q$ and $q \xrightarrow[\Delta]{a} Q$, respectively.

2.5. Reachability and projection

Formulas of the form $\Box\varphi$ and $\Diamond\varphi$ assert a one-step backwards reachability property, which we compute using a simplification of the reachability algorithm [6] due to Bouajjani et al. Cachat's extension of this algorithm to Büchi games [10] requires a technique called *projection*. Using an example, we briefly introduce the relevant techniques.

Take a PDS with the rules $p^1\, a \to p^2\, \varepsilon$ and $p^2\, b \to p^2\, ba$. The automaton A_{eg} in Fig. 2 (with q_f being the only accepting state) represents a configuration set $\mathcal{C}$. Let $Pre(\mathcal{C})$ be the set of all configurations that can reach $\mathcal{C}$ in exactly one step. To calculate $Pre(\mathcal{C})$ we first add a new set of initial states – since we do not necessarily have $\mathcal{C} \subseteq Pre(\mathcal{C})$. By applying $p^1\, a \to p^2\, \varepsilon$, any configuration of the form $\langle p^1, aw\rangle$, where w is accepted from q^2 in A_{eg}, can reach $\mathcal{C}$. Hence we add an a-transition from q^1_{new}. (Via the pop transition, we reach $\langle p^2, w\rangle \in \mathcal{L}(A_{eg})$.) Alternatively, via $p^2\, b \to p^2\, ba$, any configuration of the form $\langle p^2, bw\rangle$, where baw is accepted from q^2 in A_{eg}, can reach $\mathcal{C}$. The push, when applied backwards, replaces ba by b. We add a b-transition from q^2_{new} which skips any run over ba from q^2. Fig. 3 shows the resulting automaton.

To ensure termination of the Büchi construction, Cachat uses *projection*, which replaces a new transition to an old initial state with a transition to the corresponding new state. Hence, the transition in Fig. 3 from q^1_{new} is *replaced* by the transition

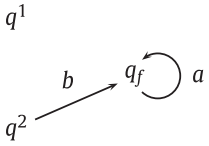

Fig. 2. The automaton A_{eg} accepting $\langle p^2, ba^*\rangle$.

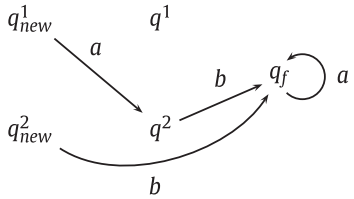

Fig. 3. A_{eg} updated by the rules $p^1\, a \to p^2\, \varepsilon$ and $p^2\, b \to p^2\, ba$.

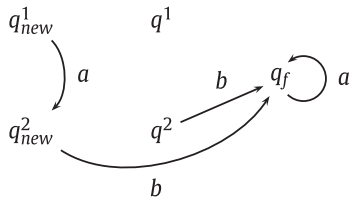

Fig. 4. The result of projecting the automaton in Fig. 3.

in Fig. 4. The old initial states are then unreachable, and deleted, which, in this case, leaves an automaton with the same states as Fig. 2 (modulo the *new* suffix) but an additional transition. In this sense, the state-set remains fixed.

3. The algorithm

The algorithm constructs an automaton representing the denotation of a given formula. Most automaton states are of the form (p, φ, c) which represents a working value of the denotation of φ restricted to the control state p. The last element c is an integer that broadly corresponds to the fixed point depth of φ in χ. We also have the states q^* and q^ε_f. The state q^* is used as a known state for accepting all stacks (of the form $\Sigma^*\bot$), and q^ε_f is the only accepting state, from which no transitions are available.

In the case of atomic propositions and bound variables, the denotation is given directly, either as a parameter to the model checking problem, or as the result of an earlier computation. For other formulas φ, we introduce new states (p, φ, c) and add transitions accordingly. For $(p, \varphi_1 \wedge \varphi_2, c)$ and $(p, \varphi_1 \wedge \varphi_2, c)$, we recursively compute states for φ_1 and φ_2 and combine the results. For $(p, \Box\varphi_1, c)$ and $(p, \Diamond\varphi_1, c)$ we compute the result for φ_1 recursively, then use a variation of the reachability techniques above to compute the result. Finally, fixed points $\sigma Z.\varphi_1$ are computed by introducing new states (p, Z, c) giving an initial value of Z, then recursively computing φ_1. The result is projected to ensure termination (as in the Büchi construction discussed above), and then assigned to be the new value of Z. This process repeats until a fixed point is reached.

Before describing the algorithm in detail, we introduce some notation. A literal $\widehat{x}$ is either x or $\neg x$ for an atomic proposition x. For a modal μ-calculus formula χ, we write $FV(\chi)$ for the set of free variables of χ. Henceforth we fix a modal μ-calculus formula χ. We shall assume χ contains no sub-formulas of the form $\sigma Z.\widehat{x}$ or $\sigma Z.X$ with $\sigma \in \{\mu, \nu\}$. Furthermore, all bound variable names are unique.

The algorithm is given in Procedures 1 to 9. Each procedure returns an automaton and a set of initial states that give the valuation of the formula it computes. These sets, I, contain a (unique) state of the form (p, φ, c) for each control state p. In general, φ is the formula whose denotation is being computed, but, in the case of a fixed point, $\varphi = Z$ where Z is the variable bound by the fixed point. Hence, we introduce the notation

$$I(p) = (p, \varphi, c) \text{ where } (p, \varphi, c) \in I$$

to denote the valuation for a given control state p. For a control state p and character a, let $Next(p, a) = \{\ (p', w)\ \mid p\, a \to p'\, w\ \}$.

We define the projection function

$$\pi_c(q) = \begin{cases} (p, \varphi, c+1) & \text{if } q = (p, \varphi, c) \\ q & \text{otherwise} \end{cases}$$

which we lift to sets of states in the obvious way. This projection function can be compared with the projections discussed in Section 2.5. Here, the states $(p, \varphi, c+1)$ correspond to the new initial states, and (p, φ, c) to the old.

For an automaton A and variable Z, we say that the variable has the set of *binding states* (p, Z, c) for all control states p such that c is the largest value for which (p, Z, c) is in A. We say an automaton A gives a valuation of an environment if it contains an initial state $(p, \widehat{x}, *)$ for every atomic proposition and control state and a binding state for every free variable and control state, such that, for a given Z, all binding states have the same c. Let $\mathcal{Q}^A_Z$ be the set of binding states of Z in A and $\mathcal{Q}^A_{\widehat{x}}$ be the set of all $(p, \widehat{x}, *)$. In addition, let $level(p, \varphi, c) = c$ and $A[\varphi/I]$ be a renaming function on automata that renames states of the form $(p, \varphi', c) \in I$ to (p, φ, c). The sets I will be suitably defined to avoid name clashes.

Procedure 1. *Denotation*$(\chi, A_V, \mathbb{P})$

Require: A pushdown system $\mathbb{P} = (\mathcal{P}, \mathcal{D}, \Sigma)$, a modal μ-calculus formula χ and an automaton A_V giving valuations for all (unbound) literals.

Ensure: A pair (A, I) such that automaton A recognises $[\![\chi]\!]^{\mathbb{P}}_V$ from initial states I.

return *Dispatch*$(A_V, \chi, 1, \mathbb{P})$

Procedure 2. $Dispatch(A, \varphi, c, \mathbb{P})$

if $\varphi = \widehat{x}$ **then**
  return $(A, \mathcal{Q}^A_{\widehat{x}})$
else if $\varphi = Z$ **then**
  return $(A, \mathcal{Q}^A_Z)$
else if $\varphi = \varphi_1 \wedge \varphi_2$ **then**
  return $And(A, \varphi_1, \varphi_2, c, \mathbb{P})$
else if $\varphi = \varphi_1 \vee \varphi_2$ **then**
  return $Or(A, \varphi_1, \varphi_2, c, \mathbb{P})$
else if $\varphi = \Box\varphi_1$ **then**
  return $Box(A, \varphi_1, c, \mathbb{P})$
else if $\varphi = \Diamond\varphi_1$ **then**
  return $Diamond(A, \varphi_1, c, \mathbb{P})$
else if $\varphi = \mu Z.\varphi_1$ **then**
  return $LFP(A, Z, \varphi_1, c, \mathbb{P})$
else if $\varphi = \nu Z.\varphi_1$ **then**
  return $GFP(A, Z, \varphi_1, c, \mathbb{P})$
end if

Procedure 3. $And(A, \varphi_1, \varphi_2, c, \mathbb{P})$

$((\mathcal{Q}_1, \Sigma, \Delta_1, \_, \mathcal{F}_1), I_1) = Dispatch(A, \varphi_1, c, \mathbb{P})$
$((\mathcal{Q}_2, \Sigma, \Delta_2, \_, \mathcal{F}_2), I_2) = Dispatch(A, \varphi_2, c, \mathbb{P})$
$A' = (\mathcal{Q}_1 \cup \mathcal{Q}_2 \cup I, \Sigma, \Delta_1 \cup \Delta_2 \cup \Delta', \_, \mathcal{F}_1 \cup \mathcal{F}_2)$
where $I = \{ (p, \varphi_1 \wedge \varphi_2, c) \mid p \in \mathcal{P} \}$
and $\Delta' = \left\{ ((p, \varphi_1 \wedge \varphi_2, c), a, Q_1 \cup Q_2) \;\middle|\; \begin{array}{c} (I_1(p), a, Q_1) \in \Delta_1 \wedge \\ (I_2(p), a, Q_2) \in \Delta_2 \end{array} \right\}$
return (A', I)

Procedure 4. $Or(A, \varphi_1, \varphi_2, c, \mathbb{P})$

$((\mathcal{Q}_1, \Sigma, \Delta_1, \_, \mathcal{F}_1), I_1) = Dispatch(A, \varphi_1, c, \mathbb{P})$
$((\mathcal{Q}_2, \Sigma, \Delta_2, \_, \mathcal{F}_2), I_2) = Dispatch(A, \varphi_2, c, \mathbb{P})$
$A' = (\mathcal{Q}_1 \cup \mathcal{Q}_2 \cup I, \Sigma, \Delta_1 \cup \Delta_2 \cup \Delta', \_, \mathcal{F}_1 \cup \mathcal{F}_2)$
where $I = \{ (p, \varphi_1 \vee \varphi_2, c) \mid p \in \mathcal{P} \}$
and $\Delta' = \left\{ ((p, \varphi_1 \vee \varphi_2, c), a, Q) \;\middle|\; \begin{array}{c} (I_1(p), a, Q) \in \Delta_1 \vee \\ (I_2(p), a, Q) \in \Delta_2 \end{array} \right\}$
return (A', I)

Procedure 5. $Box(A, \varphi_1, c, \mathbb{P})$

$((\mathcal{Q}_1, \Sigma, \Delta_1, \_, \mathcal{F}_1), I_1) = Dispatch(A, \varphi_1, c, \mathbb{P})$
$A' = (\mathcal{Q}_1 \cup I, \Sigma, \Delta_1 \cup \Delta', \_, \mathcal{F}_1)$
where $I = \{ (p, \Box\varphi_1, c) \mid p \in \mathcal{P} \}$

$$\text{and } \Delta' = \left\{ ((p, \Box\varphi_1, c), a, Q) \;\middle|\; \begin{array}{c} Next(p, a) = \{(p_1, w_1), \ldots, (p_n, w_n)\} \wedge \\ \bigwedge_{1 \le j \le n} \left(I_1(p_j) \xrightarrow[\Delta_1]{w_j} Q_j \right) \wedge \\ Q = Q_1 \cup \cdots \cup Q_n \end{array} \right\} \cup$$

$$\{ ((p, \Box\varphi_1, c), a, \{q^*\}) \mid Next(p, a) = \emptyset \wedge a \neq \bot \} \cup$$

$$\left\{ \left((p, \Box\varphi_1, c), \bot, \left\{ q^\varepsilon_f \right\} \right) \mid Next(p, \bot) = \emptyset \right\}$$

return (A', I)

We also assume that all automata have (share) the states q^* and q^ε_f, where q^ε_f is accepting and $q^* \xrightarrow{a} \{q^*\}$ for all $a \in \Sigma \setminus \{\bot\}$ and $q^* \xrightarrow{\bot} \left\{q^\varepsilon_f\right\}$. Hence, when we union state-sets and transition relations, this is not a disjoint union. Furthermore, all transitions of the form $q \xrightarrow{\bot} Q$ have $Q = \left\{q^\varepsilon_f\right\}$. Finally, we introduce a comparison operator $A \preceq A'$, which can be intuitively read as $\mathcal{L}(A) \subseteq \mathcal{L}(A')$. The precise definition is deferred to Definition 5.2.

Procedure 6. *Diamond*$(A, \varphi_1, c, \mathbb{P})$

$((\mathcal{Q}_1, \Sigma, \Delta_1, \_, \mathcal{F}_1), I_1) = \mathit{Dispatch}(A, \varphi_1, c, \mathbb{P})$
$A' = (\mathcal{Q}_1 \cup I, \Sigma, \Delta_1 \cup \Delta', \_, \mathcal{F}_1)$
where $I = \{ (p, \Diamond\varphi_1, c) \mid p \in \mathcal{P} \}$
and $\Delta' = \left\{ ((p, \Diamond\varphi_1, c), a, Q) \;\middle|\; \begin{array}{c} (p', w) \in \mathit{Next}(p, a) \wedge \\ I_1(p') \xrightarrow[\Delta_1]{w} Q \end{array} \right\}$
return (A', I)

Procedure 7. *LFP*$(A, Z, \varphi_1, c, \mathbb{P})$

$A_0 = (\mathcal{Q} \cup I_c, \Sigma, \Delta, \_, \mathcal{F})$
where $I_c = \{ (p, Z, c) \mid p \in \mathcal{P} \}$
for $i = 0$ to ω **do**
  $(B_i, I_i) = \mathit{Dispatch}(A_i, \varphi_1, c + 1, \mathbb{P})$
  $A_{i+1} = \mathit{Proj}(B_i[Z/I_i], c)$
  if $A_{i+1} \preceq A_i$ **then**
    return (A_i, I_c)
  end if
end for

Procedure 8. *GFP*$(A, Z, \varphi_1, c, \mathbb{P})$

$A_0 = (\mathcal{Q} \cup I_c, \Sigma, \Delta \cup \Delta', \_, \mathcal{F})$
where $I_c = \{ (p, Z, c) \mid p \in \mathcal{P} \}$
and Δ' contains $q \xrightarrow{a} \{q^*\}$ for all $a \neq \bot$ and $q \xrightarrow{\bot} \left\{q_f^{\varepsilon}\right\}$ for all $q \in I_c$.
for $i = 0$ to ω **do**
  $(B_i, I_i) = \mathit{Dispatch}(A_i, \varphi_1, c + 1, \mathbb{P})$
  $A_{i+1} = \mathit{Proj}(B_i[Z/I_i], c)$
  if $A_i \preceq A_{i+1}$ **then**
    return (A_i, I_c)
  end if
end for

Procedure 9. *Proj*(A, c)

$A' = A$
for all q with $\mathit{level}(q) = c + 1$ **do**
  Replace each transition $q \xrightarrow{a} Q$ in A' with $q \xrightarrow{a} \pi_c(Q)$.
end for
for all q with $\mathit{level}(q) = c$ **do**
  Remove q from A'.
end for
for all $q = (p, \varphi', c + 1)$ in A' for some p and φ' **do**
  Rename q to (p, φ', c).
end for
return A'

In Section 6 we give the pre- and post-conditions of each of the given procedures. Correctness is shown in Section 6.

4. Example

We present a fully worked example of the algorithm. Take the pushdown system presented in Example 2.1. The pushdown system has the rules

$$p \perp \to f \perp \qquad f \perp \to f \perp$$
$$p\, a \to p \qquad f\, a \to f\, a\, a$$
$$f\, a \to p\, a.$$

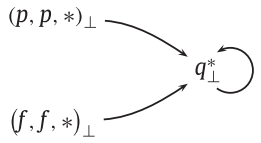

Fig. 5. The automaton giving initial valuations of p and f.

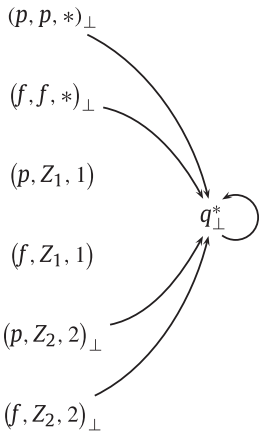

Fig. 6. The automaton after introducing initial valuations of Z_1 and Z_2.

We will evaluate the formula $\mu Z_1.\nu Z_2.\ (p \wedge \Box Z_1) \vee (f \wedge \Box Z_2)$. Initially we begin with an automaton evaluating the propositions p and f, and containing the states q_f^ε and q^* as described in Section 3. This automaton is shown in Fig. 5. For visual convenience, we have omitted the states $(p, f, *)$ and $(f, p, *)$ since they have no outgoing transitions. Also, instead of including the final state q_f^ε, we annotate each state with the subscript $\perp$ to indicate that the current state will accept on reading the bottom of stack symbol. Furthermore, since all remaining transitions are a-transitions, we will elide this label.

After fixing the initial automaton we begin to evaluate the formula. To evaluate the least fixed point of Z_1, we introduce an initial valuation of Z_1 that has no outgoing transitions. We then increment c, and evaluate the greatest fixed point of Z_2 with the initial valuation of Z_1. This begins by assigning $\mathcal{C}$ to Z_2. The automaton after these steps is shown in Fig. 6.

After creating the initial assignments to Z_1 and Z_2 we increment c again and evaluate the formula $(p \wedge \Box Z_1) \vee (f \wedge \Box Z_2)$. This recurses down the sub-formulas in turn until p, f, Z_1 or Z_2 are reached, at which point the existing valuations are used. The recursion then returns, generating states giving valuations of the formulas $\Box Z_1$, $(p \wedge \Box Z_1)$, $\Box Z_2$ and $(f \wedge \Box Z_2)$. Finally the states for $((p \wedge \Box Z_1) \vee (f \wedge \Box Z_2))$ are computed. The result is shown in Fig. 7. Alternating transitions $q \xrightarrow{a} \{q_1, q_2\}$ are illustrated using forking arrows.

At this point we have completed a recursive call of the greatest fixed point computation of Z_2. The value of $((p \wedge \Box Z_1) \vee (f \wedge \Box Z_2))$ is the new value of Z_2, so we rename these states to value Z_2 (with $c = 3$). We now perform the projections. In this case, only the transition from $(p, \Box Z_2, 3)$ to $(p, Z_2, 2)$ is affected, being replaced by a transition to $(p, Z_2, 3)$. Then we delete the old valuation of Z_2 (that is, all level 2 states) and rename all level 3 states to level 2 states. This gives a new valuation of Z_2 shown in Fig. 8.

The automaton in Fig. 8 is quite cumbersome. However, for presentational purposes, we can hide all states except those for Z_2. This is shown in Fig. 9. It is important to remember that the hidden states have not been deleted.

We now repeat the iteration, calculating the next value of Z_2. This is shown in Fig. 10. We then perform the projection, renaming and deletion operation on this automaton and repeat the iteration until the automaton remains unchanged[2] from one iteration to the next. That is, a fixed point has been reached. At this point we have concluded the first greatest fixed point computation. The result is given in Fig. 11, with some states hidden for clarity.

The result of the greatest fixed point computation gives the next value for Z_1 in the least fixed point computation. We perform the projection, renaming and deletion in the same way as the greatest fixed point case, and obtain Fig. 12. We then recompute the greatest fixed point of Z_2 with this new value of Z_1. This begins with the automaton in Fig. 13.

We repeat this greatest fixed point computation, obtaining new values of Z_1 until the least fixed point of Z_1 has been computed. Fig. 14 shows the final automaton, with the states $(p, Z_1, 1)$ and $(f, Z_1, 1)$ giving the denotation of the original formula. The reader can verify that all configurations are accepted, as required (recall, all stacks must end with the $\perp$ character). At this point we remind the reader that many states of the form $(p, \varphi, 1)$ have been omitted from the diagram because they are unreachable from $(p, Z_1, 1)$ and $(f, Z_1, 1)$, which are the interesting states in this example.

[2] This is a simplification of the termination conditions, which are given precisely in Section 5.

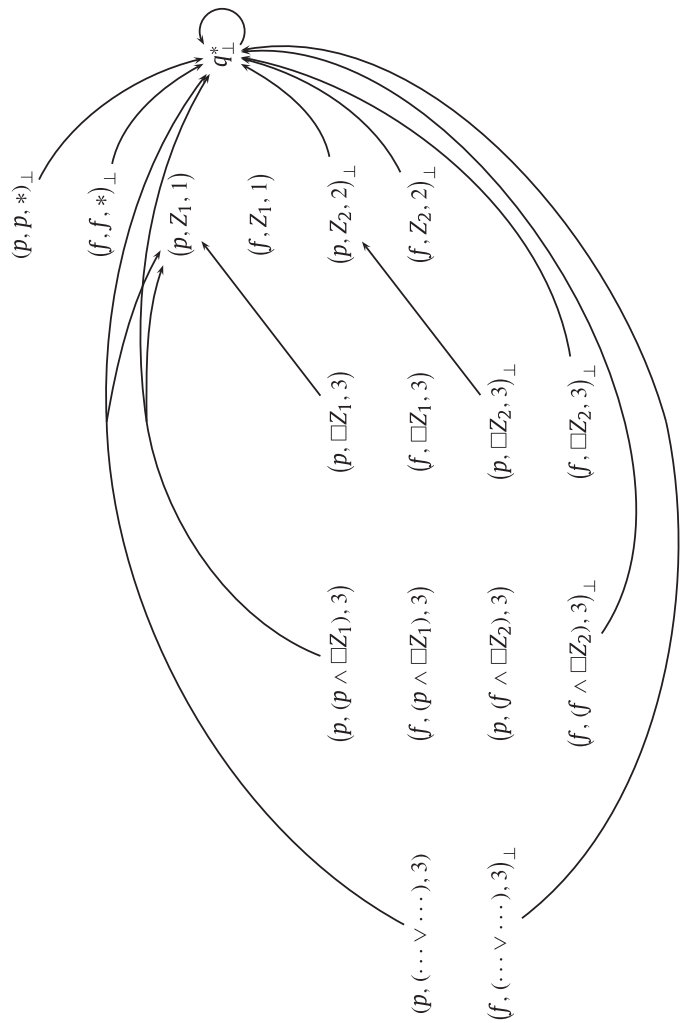

Fig. 7. The automaton after evaluating $(p \wedge \Box Z_1) \vee (f \wedge \Box Z_2)$.

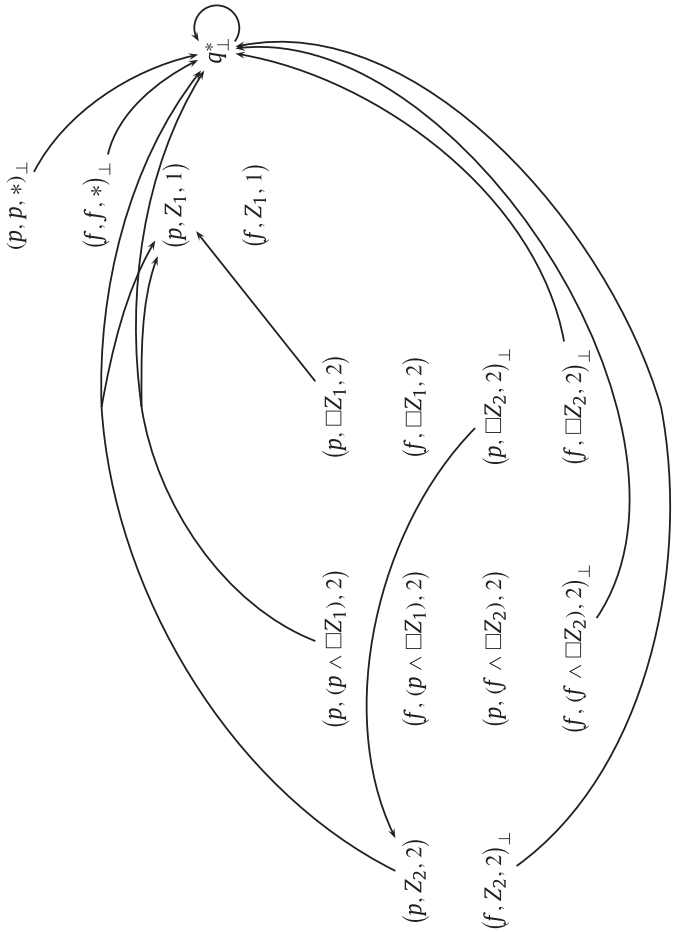

Fig. 8. The automaton after projecting, deleting and renaming.

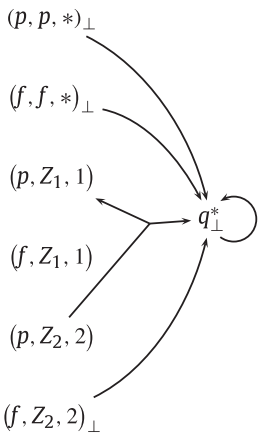

Fig. 9. The automaton in Fig. 8 with some states hidden for presentational purposes.

5. Termination

5.1. Comparing automata

We begin by defining the $\preceq$ operator described intuitively in Section 3. Observe that if we have $q \xrightarrow{a} Q$ and $q \xrightarrow{a} Q'$ with $Q \subseteq Q'$, then acceptance from Q' implies acceptance from Q. That is, the transition to Q' can, in some sense, be simulated by the transition to Q. Furthermore, acceptance from any q that is not q_f^ε implies acceptance from q^* (trivially). Using these observations, we can provide a simple test implying that $\mathcal{L}(A) \subseteq \mathcal{L}(A')$. In the following definition, $Q \ll Q'$ can be taken to mean an accepting run from Q' implies an accepting run from Q.

Definition 5.1. For all non-empty sets of states Q and Q', we define

$$Q \ll Q' \;:=\; \left((q^* \in Q \Rightarrow \exists q.q \neq q_f^\varepsilon \wedge q \in Q') \;\wedge\; (\forall q \neq q^*.q \in Q \Rightarrow q \in Q')\right).$$

One can check that $\ll$ is transitive: take $Q \ll Q'$ and $Q' \ll Q''$. If $q \neq q^* \in Q$, then $q \in Q'$ and also Q''. If $q^* \in Q$, then either $q^* \in Q'$ and the result follows from $Q' \ll Q''$, or some $q \neq q^* \in Q'$ and also $q \in Q''$, as required.

We define $\preceq$ by extending this definition to automata as follows.

Definition 5.2. For automata A and A' with state-sets $\mathcal{Q}$ and $\mathcal{Q}'$, respectively, we define $A \preceq A'$ just if for all $q \in \mathcal{Q} \cap \mathcal{Q}'$, a and Q, if $q \xrightarrow[A]{a} Q$ then for some Q', $q \xrightarrow[A']{a} Q'$ and $Q' \ll Q$.

By induction, $\preceq$ can be applied to full runs. Observe that this implies, for each shared state q, $\mathcal{L}_q(A) \subseteq \mathcal{L}_q(A')$. Since A and A' need not share the same state set, one of the consequences of using $\ll$ is that q^* can take the place of a state that is not shared between the automata. This is important after the first iteration of the greatest fixed point computations, since the recursive call may add states that were not in the initial automaton A_0.

Lemma 5.1. *For automata A and A' with state-sets $\mathcal{Q}$ and $\mathcal{Q}'$, respectively, if $A \preceq A'$ then for all $q \in \mathcal{Q} \cap \mathcal{Q}'$, w and Q, if $q \xrightarrow[A]{w} Q$ then for some Q', $q \xrightarrow[A']{w} Q'$ and $Q' \ll Q$.*

Proof. We prove for all $Q_1 \subseteq \mathcal{Q}$, $Q_2 \subseteq \mathcal{Q}'$ and w that, if $Q_2 \ll Q_1$ and $Q_1 \xrightarrow[A]{w} Q_1'$ for some Q_1', then there exists Q_2' such that $Q_2 \xrightarrow[A']{w} Q_2'$ and $Q_2' \ll Q_1'$. We proceed by induction over the length of w. When w is empty, the property is immediate. When the length is one the property holds directly from $A \preceq A'$.

Let $Q_1 = \left\{q_1^1, \ldots, q_n^1\right\}$. We have that $q_i^1 \xrightarrow[A]{a} Q_1^i$ for all $1 \leq i \leq n$ and $Q_1' = Q_1^1 \cup \cdots \cup Q_1^n$. When $a = \perp$, the property is immediate from $A \preceq A'$ and the assumed format of $\perp$-transitions. Otherwise $a \neq \perp$ and for each q_i^1 there are two cases. Either $q^* \in Q_2$ or $q_i^1 \in Q_2$. In the first case, we have $q^* \xrightarrow[A']{a} Q_2^i$ where $Q_2^i = \{q^*\}$, and hence $Q_2^i \ll Q_1^i$. In the second case, we

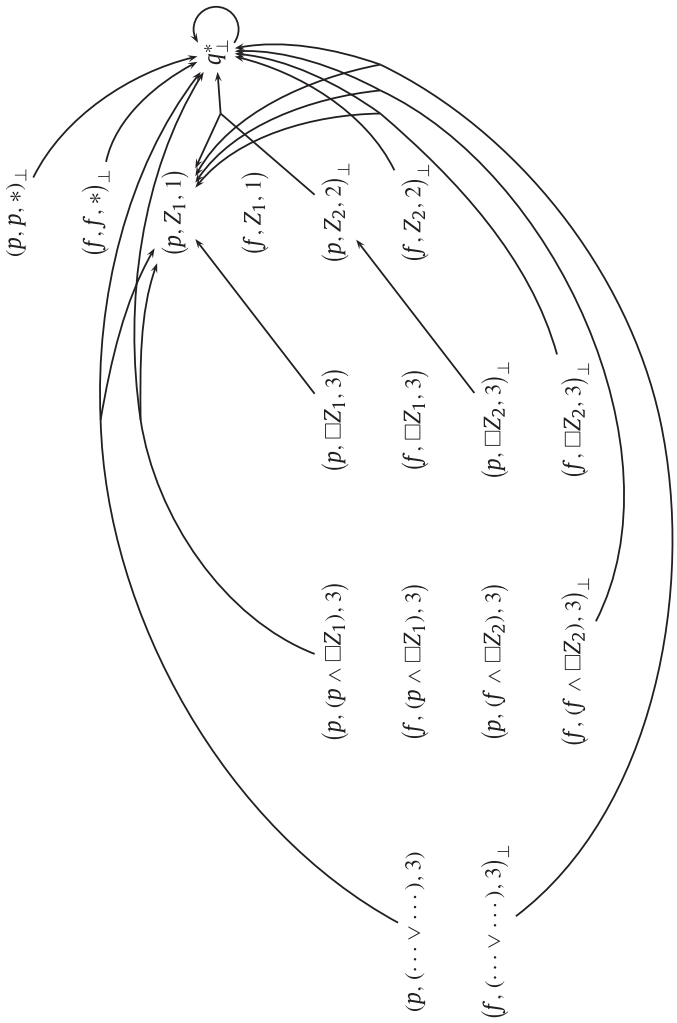

Fig. 10. The automaton after the second evaluation of $(p \wedge \Box Z_1) \vee (f \wedge \Box Z_2)$.

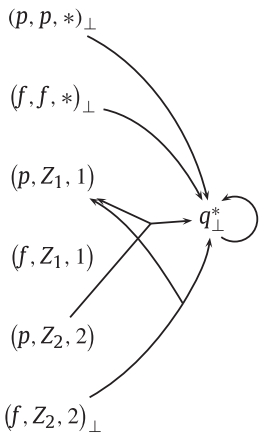

Fig. 11. The result of the first greatest fixed point calculation with some states hidden for presentational purposes.

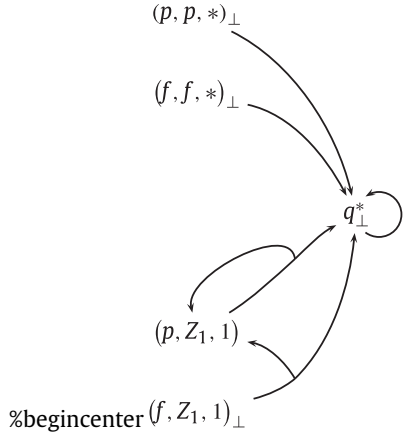

Fig. 12. The automaton after projecting, deleting and renaming for the new valuation of Z_1.

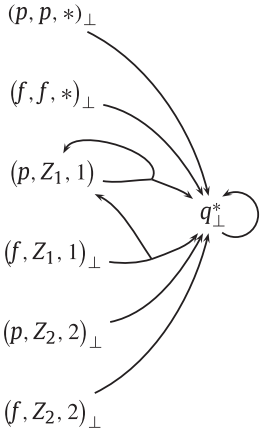

Fig. 13. The automaton before computing the next fixed point of Z_2.

have, from $A \preceq A'$ some transition $q_i^1 \xrightarrow[A']{a} Q_2^i$ with $Q_2^i \ll Q_1^i$. Thus, we have $Q_2' = Q_2^1 \cup \cdots \cup Q_2^n \ll Q_1^1 \cup \cdots \cup Q_1^n = Q_1'$ as required. This concludes the base case.

Inductively, assume $w = aw'$ and a run $Q_1 \xrightarrow[A]{a} Q_1'' \xrightarrow[A]{w'} Q_1'$. By repeating the above argument we have $Q_2 \xrightarrow[A']{a} Q_2''$ with $Q_2'' \ll Q_1''$. Then, by induction over the length of the run we have $Q_2'' \xrightarrow[A']{w'} Q_2'$ with $Q_2' \ll Q_1'$. This gives us the required run over aw. □

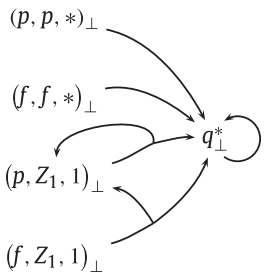

Fig. 14. The final automaton.

To prove termination, we will require the notion of an expansion.

Definition 5.3. Given an automaton A with state-set $\mathcal{Q}$, we define

$$\textsc{Expand}(A) := \left\{ q \xrightarrow{a} Q' \;\middle|\; q \xrightarrow{a} Q \text{ in } A \text{ and } Q \ll Q' \subseteq \mathcal{Q} \right\}.$$

To test termination of the fixed point computations, we compare $\textsc{Expand}(A_{i+1})$ and $\textsc{Expand}(A_i)$. In the following proofs we assume both automata share the same state-set.

Lemma 5.2. $\textsc{Expand}(A) \subseteq \textsc{Expand}(A')$ *if and only if* $A \preceq A'$.

Proof. First we assume $\textsc{Expand}(A) \subseteq \textsc{Expand}(A')$. Take $q \xrightarrow{a} Q$ in A. Then $q \xrightarrow{a} Q \in \textsc{Expand}(A)$. We have $q \xrightarrow{a} Q \in \textsc{Expand}(A')$, and therefore $q \xrightarrow{a} Q'$ is a transition of A' with $Q' \ll Q$.

In the other direction, we assume $q \xrightarrow{a} Q$ in A implies $q \xrightarrow{a} Q'$ in A'. Take $q \xrightarrow{a} Q \in \textsc{Expand}(A)$. We need $q \xrightarrow{a} Q \in \textsc{Expand}(A')$. We have some $q \xrightarrow{a} Q'$ in A with $Q' \ll Q$. Hence, we have $q \xrightarrow{a} Q''$ in A' with $Q'' \ll Q$. Hence, $q \xrightarrow{a} Q \in \textsc{Expand}(A')$ as required. □

We extend the property to runs. Hence $\textsc{Expand}(A) \subseteq \textsc{Expand}(A')$ implies $\mathcal{L}(A) \subseteq \mathcal{L}(A')$.

Lemma 5.3. *If* $\textsc{Expand}(A) \subseteq \textsc{Expand}(A')$ *then whenever* $q \xrightarrow{w} Q$ *in* A *then there is some* $Q' \ll Q$ *with* $q \xrightarrow{w} Q'$ *in* A'.

Proof. This follows directly from Lemma 5.2 and Lemma 5.1. □

5.2. Algorithm termination

We prove the following to show termination.

Lemma 5.4 (Termination)**.** *The algorithm satisfies the following properties.*

1. *Each subroutine introduces a fixed set of new states, independent of the automaton A given as input (but may depend on the other parameters). Transitions are only added to these new states.*
2. *For two input automata A and A′ (giving valuations of the same environments) such that* $A \preceq A'$*, then the returned automata* A_{out} *and* A'_{out}*, respectively, satisfy* $A'_{out} \preceq A'_{out}$*.*
3. *The algorithm terminates.*

Proof. The first of these conditions is trivially satisfied by all constructions, hence we omit the proofs. Similarly, termination is trivial for all procedures except the fixed point constructions. We will say a procedure is *monotonic* if is satisfies the second condition. The second and third conditions will be shown by mutual induction over the recursion (structure of the formula). The cases $\widehat{x}$ and Z are immediate.

Case $And(A, \varphi_1, \varphi_2, c, \mathbb{P})$:

Take the inputs $A \preceq A'$ both giving valuations for V. Let the recursive calls give us the automata $A_1 \preceq A'_1$ and $A_2 \preceq A'_2$. New transitions are only added to new states, which are the same in A_1 and A'_1 (as part of the termination conditions), and similarly for A_2 and A'_2. Let the results for the intersection be $A_\wedge$ and $A'_\wedge$, respectively. For all p we have $(p, \varphi_1 \wedge \varphi_2, c) \xrightarrow[A_\wedge]{a} Q$

derived from $I_1(p) \xrightarrow[A_1]{a} Q_1$ and $I_2(p) \xrightarrow[A_2]{a} Q_2$. Hence we have $I_1(p) \xrightarrow[A'_1]{a} Q'_1$ and $I_2(p) \xrightarrow[A'_2]{a} Q'_2$ and thus $(p, \varphi_1 \wedge \varphi_2, c) \xrightarrow[A'_\wedge]{a} Q'$ such that $Q' = Q'_1 \cup Q'_2 \ll Q_1 \cup Q_2 = Q$ as required.

Case $Or(A, \varphi_1, \varphi_2, c, \mathbb{P})$:

Take the inputs $A \preceq A'$ both giving valuations for V. Let the recursive calls give us the automata $A_1 \preceq A'_1$ and $A_2 \preceq A'_2$. New transitions are only added to new states, which are the same in A_1 and A'_1 (as part of the termination conditions), and similarly for A_2 and A'_2. Let the results for the disjunction be $A_\vee$ and $A'_\vee$, respectively. For all p we have $(p, \varphi_1 \vee \varphi_2, c) \xrightarrow[A_\vee]{a} Q$ derived from $I_1(p) \xrightarrow[A_1]{a} Q$ or $I_2(p) \xrightarrow[A_2]{a} Q$. Hence we have $I_1(p) \xrightarrow[A'_1]{a} Q'$ or $I_2(p) \xrightarrow[A'_2]{a} Q'$ and thus $(p, \varphi_1 \vee \varphi_2, c) \xrightarrow[A'_\vee]{a} Q'$ such that $Q' \ll Q$ as required.

Case $Box(A, \varphi_1, c, \mathbb{P})$:

Take the inputs $A \preceq A'$ both giving valuations for V. Let the recursive calls give us the automata $A_1 \preceq A'_1$. New transitions are only added to new states, which are the same in A_1 and A'_1 (as part of the termination conditions). Let the results for the box be $A_\Box$ and $A'_\Box$, respectively. Take a new transition $(p, \Box\varphi_1, c) \xrightarrow[A_\Box]{a} Q$. Since the case when $Next(p, a) = \emptyset$ is immediate, let $Next(p, a) = \{(p_1, w_1), \ldots, (p_n, w_n)\}$. We have $Q = Q_1 \cup \cdots \cup Q_n$ where for each $1 \leq i \leq n$ we have $I_1(p_i) \xrightarrow[A_1]{w_i} Q_i$. By $A_1 \preceq A'_1$ we have $I_1(p_i) \xrightarrow[A'_1]{w_i} Q'_i$ with $Q_i \ll Q'_i$. Hence, we have $(p, \Box\varphi_1, c) \xrightarrow[A'_\Box]{a} Q'$ with $Q' = Q'_1 \cup \cdots \cup Q'_n \ll Q_1 \cup \cdots Q_n = Q$ as required.

Case $Diamond(A, \varphi_1, c, \mathbb{P})$:

Take the inputs $A \preceq A'$ both giving valuations for V. Let the recursive calls give us $A_1 \preceq A'_1$. New transitions are only added to new states, which are the same in A_1 and A'_1 (as part of the termination conditions). Let the results for the box be $A_\Diamond$ and $A'_\Diamond$, respectively. Take a new transition $(p, \Diamond\varphi_1, c) \xrightarrow[A_\Diamond]{a} Q$. Take some $(p', w') \in Next(p, a)$. We have $I_1(p') \xrightarrow[A_1]{w'} Q$. By $A_1 \preceq A'_1$ we have $I(p') \xrightarrow[A'_1]{w'} Q'$ with $Q' \ll Q$. Hence, we have $(p, \Diamond\varphi_1, c) \xrightarrow[A'_\Diamond]{a} Q'$ with $Q' \ll Q$ as required.

Case $LFP(A, Z, \varphi_1, c, \mathbb{P})$:

Note that the state-set of A_0 is a subset of the states of A_1 (since it does not contain the states introduced by the recursive call). However, for all $i \geq 1$, all A_i have the same states. Initially we have $A_0 \preceq A_1$ since the shared states of A_0 and A_1 are either given by A (and hence have the same transitions), or have no transitions in A_0. Since the recursive call is monotonic, and the projections do not affect monotonicity, we have by induction that $A_i \preceq A_{i+1}$ for all i. For all $i \geq 1$, we have by Lemma 5.2 that $\textsc{Expand}(A_i) \subseteq \textsc{Expand}(A_{i+1})$. Since the set of states is fixed, we must eventually have $\textsc{Expand}(A_i) = \textsc{Expand}(A_{i+1})$ and hence $A_{i+1} \preceq A_i$, resulting in termination.

Monotonicity follows directly from the monotonicity of the recursive call, and that the projections do not affect the monotonicity property.

Case $GFP(A, Z, \varphi_1, c, \mathbb{P})$:

Note that the state-set of A_0 is a subset of the states of A_1 (since it does not contain the states introduced by the recursive call). However, for all $i \geq 1$, all A_i have the same states. Initially we have $A_1 \preceq A_0$ since the shared states of A_0 and A_1 are either given by A (and hence have the same transitions), or have transitions to q^* or q_f^ε that always imply $\ll$ as required. Since the recursive call is monotonic, and the projections do not affect monotonicity, we have by induction that $A_{i+1} \preceq A_i$ for all i. For all $i \geq 1$, we have by Lemma 5.2 that $\textsc{Expand}(A_{i+1}) \subseteq \textsc{Expand}(A_i)$. Since the set of states is fixed, we must eventually have $\textsc{Expand}(A_i) = \textsc{Expand}(A_{i+1})$ and hence $A_i \preceq A_{i+1}$, resulting in termination.

Monotonicity follows directly from the monotonicity of the recursive call, and that the projections do not affect the monotonicity property. □

5.3. Complexity

The algorithm runs in EXPTIME. Let m be the nesting depth of the fixed points of the formula and n be the number of states in A_V. We introduce at most $k = |\mathcal{P}| \cdot |\chi| \cdot m$ states to the automaton. Hence, there are at most $(n + k)$ states in the automaton during any stage of the algorithm. The fixed point computations iterate up to an $\mathcal{O}\left(2^{\mathcal{O}(n+k)}\right)$ number of times. Each iteration has a recursive call, which takes up to $\mathcal{O}\left(2^{\mathcal{O}(n+k)}\right)$ time. Hence the algorithm is $\mathcal{O}\left(2^{\mathcal{O}(n+k)}\right)$ overall.

6. Correctness

6.1. Valuation soundness and completeness

To prove correctness, we will introduce the notion of a *valuation profile*, which is a mapping $V : \mathcal{Q} \to \mathcal{P}(\Sigma^*\bot)$. Intuitively, a valuation profile maps each state of a automaton to a set of words that should be accepted from that state. For example, $V(q^*) = \Sigma^*\bot$ since all valid stacks are accepted from q^*. Similarly, $V(q_f^\varepsilon) = \{\varepsilon\}$. Note that we overload V to represent valuation profiles and modal μ-calculus valuations. It will be clear from the context which usage is intended.

Given a valuation profile V and some c, we can extract a modal μ-calculus valuation V_c as follows. Let $V_c(Z) = \{ \langle p, w\rangle \mid w \in V(p, Z, c') \}$ where c' is the largest $c' \leq c$ such that $V(p, Z, c')$ is defined.

We introduce *valuation soundness* and *valuation completeness* based on a profile V. We prove that all subroutines of the algorithm have this property. First, it is worth taking some time to understand the benefits of valuation soundness and completeness in proving the correctness of the algorithm.

The main challenge in proving correctness is to show that the projections do not cause any violations to correctness: the rest of the algorithm can be seen, rather straightforwardly, to be correct. Given a transition from some state $(p, \varphi, c+1)$ to a set of states Q, the effect of the projections is to replace every occurrence of (p, φ, c) in Q with $(p, \varphi, c+1)$. Valuation soundness and completeness formalises the intuition that these two states represent two working values of the same denotation. Hence, replacing one with the other will maintain correctness.

More precisely *valuation soundness* captures the observation that the existence of an a-transition in an automaton means that the a character can be pretended to any word accepted by the destination of the transition. For an automaton to be valuation sound with respect to some V, then all of its transitions must be in accordance with V.

Definition 6.1. Given a valuation V, an automaton A is V-sound just if, for all q, a and w, if A has a transition $q \xrightarrow{a} Q$ such that $w \in V(q')$ for all $q' \in Q$, then $a\,w \in V(q)$.

By induction on the length of the word, valuation soundness extends to runs of an automaton. We then obtain that all accepting runs are sound.

Lemma 6.1. *Let A be a V-sound automaton.*

1. *For all q, w and w', if A has a run $q \xrightarrow{w} Q$ such that $w' \in V(q')$ for all $q' \in Q$, then $w\,w' \in V(q)$.*
2. *For all $q \in \mathcal{Q}_A$, $\mathcal{L}_q(A) \subseteq V(q)$.*

Proof

(i) We prove by induction on the length of the word w. For the empty word, the property is trivial. When $w = a$, the property is just V-soundness. Take $w = au$ and some run $q \xrightarrow{a} Q \xrightarrow{u} Q'$ such that for all $q' \in Q'$, we have $w \in V(q')$. By the induction hypothesis, we have the property for the run $Q \xrightarrow{u} Q'$. Hence, we have for all $q' \in Q$ that, $uw' \in V(q')$. Thus, from V-soundness, we have $auw' \in V(q)$.

(ii) Take an accepting run $q \xrightarrow{w} Q_f$ of A. We have for all $q' \in Q_f = \left\{q_f^\varepsilon\right\}$, $\varepsilon \in V(q')$. Thanks to (i), we have $w \in V(q)$. □

Valuation completeness is the dual notion to valuation soundness. It says that if a character can begin a word that should be accepted from a given state, then there should be a transition that witnesses this. Furthermore, the transition should be in accordance with the given valuation V.

Definition 6.2. Given a valuation V, an automaton A is V-complete just if, for all q, a and w, if $a\,w \in V(q)$ then A has a transition $q \xrightarrow{a} Q$ such that $w \in V(q')$ for all $q' \in Q$.

By induction on the length of the word, valuation completeness extends to runs. Furthermore, an accepting run always exists when required.

Lemma 6.2. *Let A be a V-complete automaton.*

1. *For all q, w and w', if $w\,w' \in V(q)$ then A has a run $q \xrightarrow{w} Q$ such that $w' \in V(q')$ for all $q' \in Q$.*
2. *For all $q \in \mathcal{Q}_A$, $V(q) \subseteq \mathcal{L}_q(A)$.*

Proof

(i) The proof is by induction on the length of the word w. When w is empty, the property is trivial. When $w = a$, the property is simply V-completeness. Take $w = au$ and some q with $auw' \in V(q)$. From V-completeness, we have a

transition $q \xrightarrow{a} Q$ such that for all $q' \in Q$, we have $uw \in V(q')$. By induction on the length of the word, we have a run $Q \xrightarrow{u} Q'$ satisfying the property. Hence, we have $q \xrightarrow{a} Q \xrightarrow{u} Q'$ as required.

(ii) Take $w \in V(q)$. Instantiating (i) with $w' = \varepsilon$, we know A has a run $q \xrightarrow{w} Q$. Every state in Q must be accepting because ε is only accepted from accepting states and there can be no $\langle p, \varepsilon\rangle$ satisfying any denotation because ε is not a valid stack. □

6.2. Algorithm correctness

To define the correctness conditions we need to define the extension of a valuation profile by a formula φ. For a variable Z bound in φ, we denote by φ_Z the sub-formula of φ that binds Z.

Definition 6.3. Given a valuation profile V, we define V^c_φ for a given c and φ such that for sub-formulas φ' of φ

$$V^c_\varphi(p, \varphi', c') = \begin{cases} V(p, \varphi', c') & \text{if } c' < c \\ \left\{ w \;\middle|\; \langle p, w\rangle \in [\![\varphi_Z]\!]^{\mathbb{P}}_{(V^c_\varphi)_c} \right\} & \text{if } \varphi' = Z \text{ and } c' = c \\ \left\{ w \;\middle|\; \langle p, w\rangle \in [\![\varphi']\!]^{\mathbb{P}}_{(V^c_\varphi)_c} \right\} & \text{otherwise} \end{cases}.$$

(Note that this definition is circular. The definition can be made recursive by valuing the variables in order of alternation depth.)

We are now ready to state the correctness conditions.

Definition 6.4 (*Correctness conditions*)**.** The correctness conditions are as follows. Let A be the input automaton, φ be the input formula,[3] c be the input level and A' be the result.

1. We only introduce level c states.
2. If A is V-sound, A' is V^c_φ-sound.
3. If A is V-complete, A' is V^c_φ-complete.

We say that a procedure is V-sound/complete if the second/third condition is satisfied. That each procedure only introduces level c states is straightforward, hence we only show V-soundness and -completeness.

Lemma 6.3 (Valuation soundness)**.** *The algorithm is V-sound.*

Proof. The proof is by induction over the recursion. The base cases $\widehat{x}$ and Z are immediate.

Case $And(A, \varphi_1, \varphi_2, c, \mathbb{P})$:

By assumption, A is valuation sound with respect to some V. Let A_1 and A_2 be the results of the recursive calls. By induction, A_1 and A_2 are valuation sound with respect to $V^c_{\varphi_1}$ and $V^c_{\varphi_2}$, respectively.

We claim A' is sound with respect to $V^c_{\varphi_1\wedge\varphi_2}$. This only has to be shown for the new transitions $((p, \varphi_1 \wedge \varphi_2, c), a, Q_1 \cup Q_2)$ derived from $(I_1(p), a, Q_1)$ and $(I_2(p), a, Q_2)$. Suppose some w such that for all $q \in Q_1 \cup Q_2$, $w \in V^c_{\varphi_1\wedge\varphi_2}(q)$. Then, we have $w \in V^c_{\varphi_1}(q)$ and $w \in V^c_{\varphi_2}(q)$. Since A_1 and A_2 are sound, this implies $aw \in V^c_{\varphi_1}(I_1(p))$ and $aw \in V^c_{\varphi_2}(I_2(p))$ and hence $aw \in V^c_{\varphi_1\wedge\varphi_2}(p, \varphi_1 \wedge \varphi_2, c)$ as required.

Case $Or(A, \varphi_1, \varphi_2, c, \mathbb{P})$:

By assumption, A is valuation sound with respect to some V. Let A_1 and A_2 be the results of the recursive calls. By induction, A_1 and A_2 are valuation sound with respect to $V^c_{\varphi_1}$ and $V^c_{\varphi_2}$, respectively.

We claim A' is sound with respect to $V^c_{\varphi_1\vee\varphi_2}(Z, c')$. This only has to be shown for the new transitions $((p, \varphi_1 \vee \varphi_2, c), a, Q)$ derived from $(I_1(p), a, Q)$ or $(I_2(p), a, Q)$. Suppose some w such that for all $q \in Q$, $w \in V^c_{\varphi_1\vee\varphi_2}(q)$. Then, we have $w \in V^c_{\varphi_1}(q)$ or $w \in V^c_{\varphi_2}(q)$. By symmetry, we only handle the first case. Since A_1 is sound, this implies $aw \in V^c_{\varphi_1}(I_1(p))$ and hence $aw \in V^c_{\varphi_1\vee\varphi_2}(p, \varphi_1 \vee \varphi_2, c)$ as required.

Case $Box(A, \varphi_1, c, \mathbb{P})$:

We assume that A is valuation sound with respect to some valuation V. Let A_1 be the result of the recursive call. By induction A_1 is valuation sound with respect to $V^c_{\varphi_1}$. We show that A' is valuation sound with respect to $V^c_{\Box\varphi_1}$.

[3] For cases such as $And(A, \varphi_1, \varphi_2, c, \mathbb{P})$ we take, as appropriate, $\varphi = \varphi_1 \wedge \varphi_2$.

We first deal with the case when $Next(p, a) = \emptyset$. In this case, the valuation of $\Box\varphi_1$ contains all words of the form aw for some w. Hence, all added transitions are trivially sound.

Otherwise, take a new transition $((p, \Box\varphi_1, c), a, Q)$ derived from the value of $Next(p, a) = \{(p_1, w_1), \ldots, (p_n, w_n)\}$ and for all $1 \leq j \leq n$, the runs $I_1(p_j) \xrightarrow[A_1]{w_j} Q_j$, with $Q = Q_1 \cup Q_n$. Suppose for some w, $w \in V^c_{\Box\varphi_1}(q)$ for all $q \in Q$. By valuation soundness of A_1 we know $w_j w \in V^c_{\Box\varphi_1}(I_1(p_j))$ and hence, since all transitions from $\langle p, aw\rangle$ lead to configurations satisfying φ_1, $aw \in V^c_{\Box\varphi_1}(p, \Box\varphi_1, c)$ as required.

Case $Diamond(A, \varphi_1, c, \mathbb{P})$:

We assume that A is valuation sound with respect to some valuation V. Let A_1 be the result of the recursive call. By induction A_1 is valuation sound with respect to $V^c_{\varphi_1}$. We show that A' is valuation sound with respect to $V^c_{\Diamond\varphi_1}$.

Take a new transition $((p, \Diamond\varphi_1, c), a, Q)$ derived from some $(p', w') \in Next(p, a)$ and the run $I_1(p') \xrightarrow[A_1]{w'} Q$. Suppose for some w, $w \in V^c_{\Diamond\varphi_1}(q)$ for all $q \in Q$. By valuation soundness of A_1 we know $w'w \in V^c_{\Diamond\varphi_1}(I_1(p'))$ and hence, since there is a transition from $\langle p, aw\rangle$ to a configuration satisfying φ_1, $aw \in V^c_{\Diamond\varphi_1}(p, \Diamond\varphi_1, c)$ as required.

Case $LFP(A, Z, \varphi_1, c, \mathbb{P})$:

By assumption A is sound with respect to V. Let $V_\mu = V^c_{\mu Z.\varphi_1}$. Initially, A_0 is valuation sound with respect to V_μ since there are no transitions from the new states. Hence, we assume the case for A_i and prove it for A_{i+1}. By induction over the recursion, B_i is sound with respect to V_μ. Since I_i are sound with respect to φ_1 and (abusing notation) $\mu Z.\varphi_1 = \varphi_1(\mu Z.\varphi_1)$ we have that $B_i[Z/I_i]$ remains V_μ sound.

Take any transition $((p, \varphi, c), a, Q)$ in A_{i+1} and any w such that for all $q \in Q$ we have $w \in V_\mu(q)$. Consider the corresponding transition $((p, \varphi, c+1), a, Q')$ in $B_i[Z/I_i]$. All states q in Q' that are not level c or $c+1$ remain in Q, hence we have $w \in V_\mu(q)$. Furthermore, since the level c valuation of Z equals the level $c+1$ valuation, we have $w \in V_\mu(q)$ for all level c and $c+1$ states. Hence, by soundness of $B_i[Z/I_i]$ we know $aw \in V_\mu(p, \varphi, c+1)$ and therefore $aw \in V_\mu(p, \varphi, c)$ as required.

Case $GFP(A, Z, \varphi_1, c, \mathbb{P})$:

By assumption A is sound with respect to V. Let ν^α be $[\![\nu^\alpha Z.\varphi_1]\!]_{V_c}$. We begin, with a minor diversion.

Assume, A_i is valuation sound with respect to

$$V_{\alpha+1}(p, \varphi, c') = \begin{cases} V(p, \varphi, c') & \text{if } c' < c \\ \left\{ w \;\middle|\; \langle p, w\rangle \in \nu^{\alpha+1} \right\} & \text{if } \varphi = Z \text{ and } c = c' \\ \left\{ w \;\middle|\; \langle p, w\rangle \in [\![\varphi]\!]_{(V_c[Z \mapsto \nu^\alpha])} \right\} & \text{otherwise} \end{cases}.$$

We show A_{i+1} is sound with respect to

$$V_{\alpha+2}(p, \varphi, c') = \begin{cases} V(p, \varphi, c') & \text{if } c' < c \\ \left\{ w \;\middle|\; \langle p, w\rangle \in \nu^{\alpha+2} \right\} & \text{if } \varphi = Z \text{ and } c = c' \\ \left\{ w \;\middle|\; \langle p, w\rangle \in [\![\varphi]\!]_{(V_c[Z \mapsto \nu^{\alpha+1}])} \right\} & \text{otherwise} \end{cases}.$$

Let $V^{c+1}_{\alpha'} = (V_{\alpha'})^{c+1}_{\varphi_1}$. By induction, B_i is sound with respect to $V^{c+1}_{\alpha+1}$, which values Z as $\nu^{\alpha+1}$.

Take any $((p, Z, c), a, Q)$ in A_{i+1} and w such that $w \in V_{\alpha+2}(q)$. Take the corresponding transition $(I_i(p), a, Q')$ in B_i. For all $q \in Q'$ that are not level c or $c+1$ we know $q \in Q$ and hence $w \in V_{\alpha+2}(q)$ which is a subset of $V_{\alpha+1}(q)$. For level c states the same subset argument holds. For level $c+1$ the valuations are the same. Hence, the pre-conditions for the soundness condition are satisfied, and from the soundness of B_i we know $aw \in V^{c+1}_{\alpha+1}(I_i(p)) = \mu^{\alpha+2} = V_{\alpha+2}(p, Z, c)$, as required.

Take any $((p, \varphi, c), a, Q)$ with $\varphi \neq Z$ in A_{i+1} and w such that $w \in V_{\alpha+2}(q)$. Take the corresponding transition $((p, \varphi, c+1), a, Q')$ in B_i. For all $q \in Q'$ that are not level c or $c+1$ we know $q \in Q$ and hence $w \in V_{\alpha+2}(q)$ which is a subset of $V_{\alpha+1}(q)$. For level c states the same subset argument holds. For level $c+1$ the valuations are the same. Hence, the pre-conditions for the soundness condition are satisfied, and from the soundness of B_i we know $aw \in V^{c+1}_{\alpha+1}(p, \varphi, c+1) = V_{\alpha+2}(p, \varphi, c)$, as required.

Thus, A_{i+1} is sound with respect to $V_{\alpha+2}$ as required.

We are now ready to prove the main result by induction over the ordinals. We have that, $A' = A_i = A_{i+1}$. A_0 is trivially sound with respect to V_0. Then, by the argument above, A_i is sound with respect to V_i. The case of a successor ordinal also follows from the above. For a limit ordinal λ, we have soundness for all $\alpha < \lambda$. Since $\theta^\lambda = \bigcap_{\alpha<\lambda} \theta^\alpha$, the result follows because each configuration in the limit appears in all smaller approximants, and we are sound for all smaller approximants (and trivially for the zeroth approximant). To regain the induction hypothesis for successor ordinals, we simply apply the successor construction once, which keeps all (p, φ, c) where $\varphi \neq Z$ sound for the limit, while (p, Z, c) becomes sound for $\nu^{\lambda+1}$. □

Lemma 6.4 (Valuation completeness). *The algorithm is V-complete.*

Proof. The proof is by induction over the recursion. The base cases $\widehat{x}$ and Z are immediate.

Case $And(A, \varphi_1, \varphi_2, c, \mathbb{P})$:

By assumption, A is valuation complete with respect to some V. Let A_1 and A_2 be the results of the recursive calls. By induction, A_1 and A_2 are valuation complete with respect to $V^c_{\varphi_1}$ and $V^c_{\varphi_2}$, respectively. We have $V^c_{\varphi_1\wedge\varphi_2}$ as above. We claim A' is complete with respect to this valuation. This only has to be shown for the new states of the form $q_{new} = (p, \varphi_1 \wedge \varphi_2, c)$. Suppose $aw \in V^c_{\varphi_1\wedge\varphi_2}(q_{new})$. This implies $aw \in V^c_{\varphi_1}(I_1(p))$ and $aw \in V^c_{\varphi_2}(I_2(p))$. Since A_1 and A_2 are valuation complete, we have some transitions $(I_1(p), a, Q_1)$ and $(I_2(p), a, Q_2)$ such that for all $q \in Q_1 \cup Q_2$, $w \in V^c_{\varphi_1\wedge\varphi_2}(q)$. This implies the transition $((p, \varphi_1 \wedge \varphi_2, c), a, Q_1 \cup Q_2)$ is in A'. This transition witnesses completeness.

Case $Or(A, \varphi_1, \varphi_2, c, \mathbb{P})$:

By assumption, A is valuation complete with respect to some V. Let A_1 and A_2 be the results of the recursive calls. By induction, A_1 and A_2 are valuation complete with respect to $V^c_{\varphi_1}$ and $V^c_{\varphi_2}$, respectively. Take $V^c_{\varphi_1\vee\varphi_2}$ as above. We claim A' is complete with respect to this valuation. This only has to be shown for the new states of the form $q_{new} = (p, \varphi_1 \vee \varphi_2, c)$. Suppose $aw \in V^c_{\varphi_1\vee\varphi_2}(q_{new})$. This implies $aw \in V^c_{\varphi_1}(I_1(p))$ or $aw \in V^c_{\varphi_2}(I_2(p))$. We assume the first case by symmetry. Since A_1 is valuation complete, we have some transition $(I_1(p), a, Q)$ such that for all $q \in Q$, $w \in V^c_{\varphi_1}(q)$. This implies the transition $((p, \varphi_1 \vee \varphi_2, c), a, Q)$ is in A'. This transition witnesses completeness.

Case $Box(A, \varphi_1, c, \mathbb{P})$:

We are given that A is valuation complete with respect to some valuation V. Let A_1 be the result of the recursive call. By induction we have completeness of A_1 with respect to $V^c_{\varphi_1}$. We show A' is complete with respect to $V^c_{\Box\varphi_1}$.

In the case that $Next(p, a) = \emptyset$, we either have $a = \bot$ and the transition from $(p, \Box\varphi_1, c)$ to $\left\{q^{\varepsilon}_f\right\}$ witnesses completeness, or we have $a \neq \bot$ and the transition from $(p, \Box\varphi_1, c)$ to $\{q^*\}$ witnesses completeness.

Otherwise, assume we have aw such that $aw \in V^c_{\Box\varphi_1}(p, \Box\varphi_1, c)$ and $Next(p, a) = \{(p_1, w_1), \ldots, (p_n, w_n)\}$. Hence, for all $1 \leq j \leq n$, we have $w_j w \in V^c_{\Box\varphi_1}(I_1(p_j))$. By completeness of A_1 we have runs $I_1(p_j) \xrightarrow[A_1]{w_j} Q_j$ such that for all $q \in Q_j$, $w \in V^c_{\Box\varphi_1}(q)$. Hence, the transition $((p, \Box\varphi_1, c), a, Q_1 \cup \cdots \cup Q_n)$ witnesses completeness.

Case $Diamond(A, \varphi_1, c, \mathbb{P})$:

We are given that A is valuation complete with respect to some valuation V. Let A_1 be the result of the recursive call. By induction we have completeness of A_1 with respect to $V^c_{\varphi_1}$. We show A' is complete with respect to $V^c_{\Diamond\varphi_1}$.

Assume some aw such that $aw \in V^c_{\Diamond\varphi_1}(p, \Diamond\varphi_1, c)$ and take $(p', w') \in Next(p, a)$ such that we have $\langle p', w'w\rangle \in V^c_{\Diamond\varphi_1}(I_1(p'))$. By completeness of A_1 we have a run $I_1(p') \xrightarrow[A_1]{w'} Q$ such that for all $q \in Q$, $w \in V^c_{\Diamond\varphi_1}(q)$. Hence, the transition $((p, \Diamond\varphi_1, c), a, Q)$ witnesses completeness.

Case $LFP(A, Z, \varphi_1, c, \mathbb{P})$:

By assumption A is complete with respect to V. Let μ^α be $[\![\mu^\alpha Z.\varphi_1]\!]_{V_c}$. We begin, as before, with a minor diversion.

Assume, A_i is valuation complete with respect to

$$V_{\alpha+1}(p, \varphi, c') = \begin{cases} V(p, \varphi, c') & \text{if } c' < c \\ \left\{ w \;\middle|\; \langle p, w\rangle \in \mu^{\alpha+1} \right\} & \text{if } \varphi = Z \text{ and } c = c' \\ \{ w \mid \langle p, w\rangle \in [\![\varphi]\!]_{(V_c[Z\mapsto\mu^\alpha])} \} & \text{otherwise} \end{cases}.$$

We show A_{i+2} is complete with respect to

$$V_{\alpha+2}(p, \varphi, c') = \begin{cases} V(p, \varphi, c') & \text{if } c' < c \\ \left\{ w \;\middle|\; \langle p, w\rangle \in \mu^{\alpha+2} \right\} & \text{if } \varphi = Z \text{ and } c = c' \\ \left\{ w \;\middle|\; \langle p, w\rangle \in [\![\varphi]\!]_{(V_c[Z\mapsto\mu^{\alpha+1}])} \right\} & \text{otherwise} \end{cases}.$$

Let $V^{c+1}_{\alpha'} = (V_{\alpha'})^{c+1}_{\varphi_1}$. By induction, B_i is complete with respect to $V^{c+1}_{\alpha+1}$, which values Z as $\mu^{\alpha+1}$.

For each (p, Z, c) in A_{i+1}, take some $aw \in V_{\alpha+2}(p, Z, c) = \mu^{\alpha+2}$. Since $\mu^{\alpha+2} = \varphi(\mu^{\alpha+1})$ we have that $aw \in V^{c+1}_{\alpha+1}(I_i(p))$ from the completeness of B_i. Hence there was a complete transition $(I_i(p), a, Q)$ in B_i. For all states $q \in Q$ not of level c or $c + 1$, the completeness conditions remain satisfied after the projections in A_{i+1}. For level c state (p', φ, c) we know that

$w \in V_{\alpha+1}(p', \varphi, c)$ which is a subset of $V_{\alpha+2}(p', \varphi, c)$ and we are done. For a level $c+1$ state $(p', \varphi, c+1)$ we know $w \in V^{c+1}_{\alpha+1}(p', \varphi, c+1)$ which is also a subset of $V_{\alpha+2}(p', \varphi, c)$, hence we are done.

For each (p, φ, c) in A_{i+1} with $\varphi \neq Z$, take some $aw \in V_{\alpha+2}(p, \varphi, c) = \varphi_1(\mu^{\alpha+1})$. From the completeness of B_i there was a complete transition $((p, \varphi, c+1), a, Q)$ in B_i. For all states $q \in Q$ not of level c or $c+1$, the completeness conditions remain satisfied after the projections in A_{i+1}. For a level c state (p', φ', c) we know that $w \in V_{\alpha+1}(p', \varphi', c)$ which is a subset of $V_{\alpha+2}(p', \varphi', c)$ and we are done. For a level $c+1$ state $(p', \varphi', c+1)$ we know $w \in V^{c+1}_{\alpha+1}(p', \varphi', c+1)$ which is also a subset of $V_{\alpha+2}(p', \varphi', c)$, hence we are done.

Thus, A_{i+1} is complete with respect to $V_{\alpha+2}$ as required.

We are now ready to prove the main result by induction over the ordinals. Trivially, $A' = A_i = A_{i+1}$ (for some $i \geq 1$) is sound with respect to V_0. This is because A_0 is complete with respect to the extension of V mapping Z to μ^0, and the recursive call ensures completeness with respect to the full V_0. The case of a successor ordinal was shown above. For a limit ordinal λ, we have completeness for V_α for all $\alpha < \lambda$. Since $\mu^\lambda = \bigcup_{\alpha<\lambda} \mu^\alpha$, the result follows because each configuration in the limit appears in some smaller approximant, and the transition witnessing completeness for the approximant witnesses completeness for the limit. To regain the induction hypothesis for successor ordinals, we simply apply the successor construction once, which keeps all (p, φ, c) where $\varphi \neq Z$ complete for the limit, while (p, Z, c) becomes complete for $\mu^{\lambda+1}$.

Case $GFP(A, Z, \varphi_1, c, \mathbb{P})$:

By assumption A is complete with respect to V. Initially, A_0 is valuation complete with respect to the extension of V that values Z as $[\![\nu Z.\varphi_1]\!]_{V_c}$. After the first iteration, using a specialisation of the argument below, we have that A_1 is complete with respect to $V^c_{\nu Z.\varphi_1}$, which we will abbreviate as V_ν.

We assume completeness with respect to V_ν for A_i and prove it for A_{i+1}. By induction over the recursion, B_i is complete with respect to V_ν. Since I_i are complete and the denotation of $\nu Z.\varphi_1$ is always equal to the denotation of φ_1 with the value of Z set to $\nu Z.\varphi_1$ we have that $B_i[Z/I_i]$ remains V_ν complete.

Take any $aw \in V_\nu(p, \varphi, c)$. Since $V_\nu(p, \varphi, c) = V_\nu(p, \varphi, c+1)$ we have a transition $((p, \varphi, c+1), a, Q)$ in $B_i[Z/I_i]$ that witnesses completeness for $B_i[Z/I_i]$. From this transition we have $((p, \varphi, c), a, \pi_c(Q))$ in A_{i+1}. For all $q \in Q$ of level less than c we have from B_i that $w \in V_\nu(q)$. For q of level c and $c+1$ we have $w \in V_\nu(\pi_c(q))$ from $V_\nu(p', \varphi', c) = V_\nu(p', \varphi', c+1)$ for all p' and φ'. Hence we have a transition witnessing completeness, as required. □

7. Termination and correctness of *Denotation*$(\chi, A_V, \mathbb{P})$

Termination and valuation soundness and completeness for the called subroutines are given in Lemma 5.4, Lemma 6.3 and Lemma 6.4.

Theorem 7.1. *Let* $(A, I) = Denotation(\chi, A_V, \mathbb{P})$ *where* A_V *describes a valuation* V*. The states* I *of* A *give the denotation* $[\![\chi]\!]^{\mathbb{P}}_V$.

Proof. Observe that A_V is automatically V-sound and -complete. There are two cases when χ is not $\widehat{x}$ or Z. Either $I = \{ (p, Z, 1) \mid p \in \mathcal{P} \}$ when $\chi = \sigma Z.\varphi(Z)$ for $\sigma \in \{\mu, \nu\}$, or $I = \{ (p, \chi, 1) \mid p \in \mathcal{P} \}$ otherwise. In both cases, from Lemma 6.3 with Lemma 6.1 and Lemma 6.4 with Lemma 6.2 we have the theorem as required. □

8. Application to parity games

We have described a new algorithm for computing directly the denotation of a modal μ-calculus formula χ over a pushdown system. This is an extension of a parity games algorithm presented at CONCUR [16]. The parity games algorithm takes advantage of the following modal μ-calculus description – appearing in Walukiewicz' 1996 paper [23][4] – of Éloïse's winning regions of a pushdown parity game $\mathcal{G}$.

$$\mathcal{W}_E = [\![\mu Z_1.\nu Z_2. \ldots \mu Z_{m-1}.\nu Z_m.\varphi_E(Z_1, \ldots, Z_m)]\!]^{\mathcal{G}}_V$$

where m is the maximum parity (assumed even), V is a valuation of the variables,[5] and

$$\varphi_E(Z_1, \ldots, Z_m) := \left(E \Rightarrow \bigwedge_{c \in \{1,\ldots,m\}} (c \Rightarrow \Diamond Z_c) \right) \wedge \left(\neg E \Rightarrow \bigwedge_{c \in \{1,\ldots,m\}} (c \Rightarrow \Box Z_c) \right)$$

where E is an atomic proposition asserting the current configuration is Éloïse's and, for $1 \leq c \leq m$, c asserts that the priority of the current control state is c.

Hence, we obtain an algorithm for computing Éloïse's winning regions of a pushdown parity game as a corollary of our main result. However, a naive application of the algorithm presented here will introduce many extra intermediate states in

[4] Equivalent characterisations are presented by, e.g., Arnold and Niwinski [2].

[5] The valuation is initially empty since the formula has no free variables.

Procedure 10. *PhiE*$(A, \varphi_1, c, \mathbb{P})$

Let $(\mathcal{Q}_1, \Sigma, \Delta_1, \_, \mathcal{F}_1) = A$
and $I_k = \mathcal{Q}_{Z_k}$ for all $1 \leq k \leq m$ in
$A' = (\mathcal{Q}_1 \cup I, \Sigma, \Delta_1 \cup \Delta', \_, \mathcal{F}_1)$
where $I = \{ (p, \varphi_E, c) \mid p \in \mathcal{P} \}$ and

$$\Delta' = \left\{ ((p, \varphi_E, c), a, Q) \;\middle|\; \begin{array}{c} Eloise(p) \wedge \\ (p', w) \in Next(p, a) \wedge \\ I_{\Omega(p)}(p') \xrightarrow[\Delta_1]{w} Q \end{array} \right\} \cup$$

$$\left\{ ((p, \varphi_E, c), a, Q) \;\middle|\; \begin{array}{c} Abelard(p) \wedge \\ Next(p, a) = \{(p_1, w_1), \ldots, (p_n, w_n)\} \wedge \\ \bigwedge\limits_{1 \leq j \leq n} \left(I_{\Omega(p)}(p_j) \xrightarrow[\Delta_1]{w_j} Q_j \right) \wedge \\ Q = Q_1 \cup \cdots \cup Q_n \end{array} \right\} \cup$$

$$\{ ((p, \varphi_E, c), a, \{q^*\}) \mid Abelard(p) \wedge Next(p, a) = \emptyset \wedge a \neq \bot \} \cup$$

$$\left\{ \left((p, \varphi_E, c), \bot, \left\{ q_f^{\varepsilon} \right\} \right) \mid Abelard(p) \wedge Next(p, \bot) = \emptyset \right\}$$

return (A', I)

computing φ_E. The algorithm presented in Concur can be seen as an optimised version of the modal μ-calculus algorithm for the special case of parity games. Conversely, our direct algorithm for modal μ-calculus can be expected to require fewer states than a reduction to parity games followed by an application of the parity games algorithm. However, it is possible that these additional states do not have a significant effect on performance (they may be effectively ignored during the computation). Hence, we show experimentally in the next section that each algorithm outperforms the other on the tasks they are designed for.

To obtain the parity games algorithm presented at Concur we compute the fixed points as presented in Section 3. However, we replace the computation of $\varphi_E(Z_1, \ldots, Z_m)$ with the algorithm in Procedure 10 where $\Omega : \mathcal{P} - > \{1, \ldots, m\}$ assigns colours to control states and *Eloise*(p) holds whenever Éloïse owns p and analogously for *Abelard*(p). In essence, Procedure 10 simply evaluates $\Box Z_{\Omega(p)}$ on all states p owned by Abelard, and $\Diamond Z_{\Omega(p)}$ when the state is owned by Éloïse. The required correctness and monotonicity proofs for Procedure 10 are straightforward and omitted.

9. Experimental results

We constructed a prototypical explicit state implementation of the algorithm described here for computing the denotation of a modal μ-calculus formula for a given PDS. We have also implemented the winning regions construction presented at Concur [16]. Both algorithms are implemented in OCaml.

In this paper, we intend only to analyse the behaviour of the algorithms over generic pushdown systems, rather than over those tied to a particular application. Separately, we have applied the algorithm to dataflow analysis of Java programs. The results of this work is due to be reported in SPIN [17].

We compared the two algorithms both on randomly generated modal μ-calculus problems and on randomly generated games. We discuss the common features of these tests here, and give, in the sections that follow, information specific to analysing formulas and analysing games. In both cases, each PDS was of size n, ranging from 5 to 150. Each generated PDS had n states and n characters. The number of transitions ranged between n^2 and $2n^2$ and were, with equal probability, of the form $p\,a \to p'\,w$ where the length of w was either 0, 1 or 2.

We ran the experiments on a 1 GHz AMD Dual Core with 4 Gb of RAM. A timeout of 10 min was set. In the cases where both approaches succeeded, we compared the performance both in terms of runtime and the maximum number of transitions of the multi-automaton at any point during the computation. Both metrics were used because the runtime may be sensitive to particular implementation details such as data structures. We computed the percentage differences in the figures as follows, where μ represents the value for the denotation approach, and g the value for the game approach,

$$100 \times \frac{\mu - g}{(\mu + g)/2}.$$

That is, the difference is given as a percentage of the average of the two values. Since we did not take the absolute difference, a negative value indicates that the denotation value was the lowest (best), and a positive value vice versa. Note that, using this approach, the maximum percentage difference is $\pm 200\%$.

We then analysed and plotted the data using the statistical package *R*. Violin plots of the data are shown in Fig. 15. The width of a violin plot indicates the (relative) number of data points for the appropriate percentage difference. Hence, we can

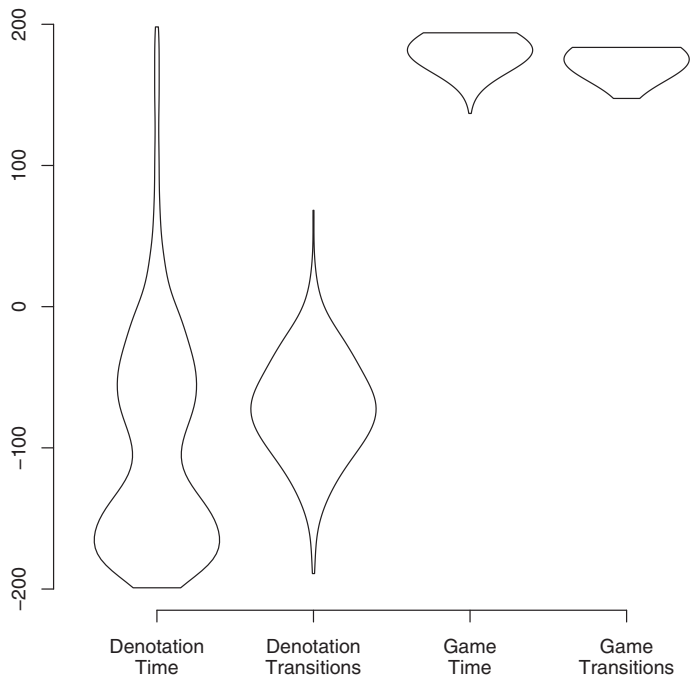

Fig. 15. The percentage difference in the performance of the two algorithms for computing the denotation of a formula and the winning region of a game, both in terms of time and maximum number of transitions. A negative value indicates that the denotation approach was superior, and a positive value favours the game approach. The label indicates whether the problem instance was a modal μ-calculus formula, or a parity game. Each violin plot compares both approaches for the stated problem and performance metric.

immediately see that, in general, the denotation approach outperformed the game approach for computing the denotation of a modal μ-calculus formula, whilst the game approach works best for constructing the winning region of a parity game.

9.1. Analysing formulas

We generated 1352 pairs of PDS and modal μ-calculus formulas and compared the denotation approach with the game approach for evaluating the formula. Each modal μ-calculus formula had a maximum connective depth of 5, a minimum fixed point depth of 2, lengths between 6 and 23 and up to 10 propositions, with each proposition having a 10% probability of holding at a given pair of control state and top of stack character. Furthermore, we insisted that each bound variable occurred within at least one $\Box$ or $\Diamond$ operator.

When beginning with a modal μ-calculus instance, there were 184 instances where both algorithms failed, 41 instances where the denotation approach failed only, and 49 where only the game approach failed. For successful runs, the mean percentage time difference was -96 (with 95% confidence interval $[-100, -91]$) and the transition difference was -71 (CI $[-73, -69]$).

When building the denotation of a modal μ-calculus formula, although the direct denotation approach did outperform the game approach in most cases, it did not do so as reliably as the game approach outperformed the denotation approach when analysing games. We expect that this is because the structure of a formula can vary a lot more than the structure of a game. Furthermore, the translation from formula to game may have the effect of optimising the computation in two ways. Firstly, identical sub-formulas will be identified, avoiding repeated computation. Secondly, if, for example, a formula contains two nested fixed points of the same kind (that is, the fixed points do not alternate), then the resulting game will have one priority representing both fixed points. Hence, only a single fixed point iteration will be required. At present, the denotation approach will perform two nested iterations, one for each fixed point. Both of these observations suggest possibilities for future optimisation.

9.2. Analysing games

We generated 1391 pushdown parity games and compared the two approaches for computing Éloïse's winning region of the game. Each game had either 2 or 3 colours, and each control state had an equal chance of belonging to Éloïse or Abelard.

When beginning with a parity game, the game approach completed in all cases, but the denotation approach failed 184 times. In successful cases, the mean percentage time difference was 182 (CI [181, 182]) and the transition difference was 175 (CI [175, 175]). All results are rounded to zero decimal places.

The spread of the differences is tightest when constructing the winning region of a parity game rather than when analysing formulas. This is likely to be because both the game approach and the denotation approach evaluate the formula $\mu Z_1.\nu Z_2.\ldots.\mu Z_{m-1}.\nu Z_m.\varphi_E(Z_1,\ldots,Z_m)$. However, whilst the game approach optimises the computation of $\varphi_E(Z_1,\ldots,Z_m)$ into one step, the denotation approach computes each sub-formula in turn. Hence, a predictable slow down is to be expected.

10. Conclusion and future work

We have presented a direct algorithm for computing the denotation of a modal μ-calculus formula over a given pushdown system. Further this generalises previous work presented in Concur 2009 which gives an algorithm for computing the winning regions of a pushdown parity game [16]. Although the two problems are inter-reducible, we have shown experimentally that each algorithm is better at solving the problem for which it was designed. Hence the two algorithms complement each other. Conditions such as fairness that are naturally expressed as parity conditions can be evaluated most effectively with the parity games algorithm, whilst general modal μ-calculus properties are better evaluated using the denotation approach.

In the case of parity games, we would like to be able to compute, in addition to the winning regions, the winning strategies of a given game. Analogously, when computing the denotation of a formula, we would also like to be able to generate proof trees. These are pressing avenues of future work.

The challenge is to apply our implementation to real world examples. A symbolic implementation would be desirable, but it is unclear how to combine a symbolic representation of the pushdown system with the use of alternating multi-automata. A complementary approach is to use an abstraction-refinement loop to minimise the size of the pushdown systems.

Acknowledgments

This work is supported by EPSRC (EP/F036361). We are greatly indebted to Arnaud Carayol for his invaluable assistance.

References

[1] R. Alur, S. Chaudhuri, P. Madhusudan, A fixpoint calculus for local and global program flows, in: in: POPL '06: Conference Record of the 33rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages,, ACM Press, New York, NY, USA, 2006, pp. 153–165.
[2] A. Arnold, D. Niwiński, Rudiments of μ-Calculus, Elsevier, Amsterdam, The Netherlands, 2001.
[3] T. Ball, S.K. Rajamani, Bebop: a symbolic model checker for boolean programs, in: in: Proceedings of the Seventh International SPIN Workshop on SPIN Model Checking and Software Verification, Springer-Verlag, London, UK, 2000, pp. 113–130.
[4] T. Ball, S.K. Rajamani, The SLAM project: debugging system software via static analysis, in: Conference Record of POPL'02: The 29th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Portland, Oregon, January 16–18, 2002, pp. 1–3.
[5] R.V. Book, F. Otto, String-Rewriting Systems, Springer-Verlag, 1993.
[6] A. Bouajjani, J. Esparza, O. Maler, Reachability analysis of pushdown automata: application to model-checking, in: International Conference on Concurrency Theory, 1997, pp. 135–150.
[7] J.C. Bradfield, C.P. Stirling, Modal logics and mu-calculi: an introduction, in: Handbook of Process Algebra, 2001, pp. 293–330.
[8] O. Burkart, B. Steffen, Composition, decomposition and model checking of pushdown processes, Nord. J. Comput. 2 (2) (1995) 89–125.
[9] O. Burkart, B. Steffen, Model checking the full modal mu-calculus for infinite sequential processes, Theor. Comput. Sci. 221 (1–2) (1999) 251–270.
[10] T. Cachat, Symbolic strategy synthesis for games on pushdown graphs, in: in: ICALP '02: Proceedings of the 29th International Colloquium on Automata, Languages and Programming, Springer-Verlag, London (2002) 704–715.
[11] T. Cachat, Games on Pushdown Graphs and Extensions, Ph.D. Thesis, RWTH Aachen, 2003.
[12] E.A. Emerson, C.S. Jutla, Tree automata, mu-calculus and determinacy, in: in: SFCS '91: Proceedings of the 32nd Annual Symposium on Foundations of Computer Science, Springer, Washington, DC, USA (1991) 368–377.
[13] J. Esparza, A. Kučera, S. Schwoon, Model-checking LTL with regular valuations for pushdown systems, in: Proceedings of the TACS 2001, Lecture Notes in Computer Science vol. 2215 (2001) 306–339.
[14] K. Etessami, Analysis of recursive game graphs using data flow equations, in: VMCAI (2004) 282–296.
[15] A. Finkel, B. Willems, P. Wolper, A direct symbolic approach to model checking pushdown systems, in: in: Proceedings of the Second International Workshop on Verification of Infinite State Systems (INFINITY '97), Bologna, Italy, July 11–12, 199.7, Electronic Notes in Theoretical Computer Science vol. 9, Elsevier (1997).
[16] M. Hague, C.-H.L. Ong, Winning regions of pushdown parity games: a saturation method, in: in: Concurrency Theory, Proceedings of 20th International Conference, CONCUR 2009, Bologna, Italy, September 1–4, 2009, LNCS vol. 5710, Springer-Verlag (2009) 384–398.
[17] M. Hague, C.-H.L. Ong, Analysing mu-calculus properties of pushdown systems, SPIN, in press.
[18] N.D. Jones, S.S. Muchnick, Even simple programs are hard to analyze, in: in: POPL '75: Proceedings of the Second ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages, ACM, New York, NY, USA (1975) 106–118.
[19] N. Piterman, M.Y. Vardi, Global model-checking of infinite-state systems, in: CAV (2004) 387–400.
[20] T. Reps, S. Schwoon, S. Jha, D. Melski, Weighted pushdown systems and their application to interprocedural dataflow analysis, Sci. Comput. Program. 58 (1–2) (2005) 206–263.
[21] S. Schwoon, Model-checking Pushdown Systems, Ph.D. Thesis, Technical University of Munich, 2002.
[22] O. Serre, Note on winning positions on pushdown games with ω-regular conditions, Inform. Process. Lett. 85 (2003) 285–291.
[23] I. Walukiewicz, Pushdown processes: games and model checking, in: Rajeev Alur, Thomas A. Henzinger (Eds.), Proceedings of the Eighth International Conference on Computer Aided Verification CAV, vol. 1102, Springer-Verlag, New Brunswick, NJ, USA, 1996, pp. 62–74.